TURING 图灵程序设计丛书

（第3版）

Stylin' with CSS
A Designer's Guide
Third Edition

CSS设计指南

[英] Charles Wyke-Smith 著
李松峰 译

- 著译双馨，CSS设计入门首选
- 亚马逊读者好评如潮
- 涵盖HTML5、CSS3和响应式设计

New Riders



人民邮电出版社
POSTS & TELECOM PRESS



Charles Wyke-Smith

一直从事媒体相关工作。20世纪80年代中期，他与人在旧金山共同创办了PRINTZ Electronic Design，当时这可是一家全部工作都使用计算机处理的设计工作室。另外，他曾在富国银行、ESPN电子游戏担任领导和顾问，在Benefitfocus公司担任过用户体验总监。2009年，他与人合办了PeopleMatter公司，提供人力资源服务。目前，他是一家新的创业公司Bublish的CEO，这家公司提供图书策划等方面的服务。

Charles是一位乐手和多本Web开发图书的作者，除了本书，他还著有*Codin' for the Web: A Designer's Guide to Developing Dynamic Web Sites*、*Scriptin' with JavaScript and AJAX: A Designer's Guide*和*Visual Stylin' with CSS3*。他和妻子，还有两个女儿住在美国南卡罗来纳州查尔斯顿市。



李松峰

图灵公司QA部主任。2006年起投身翻译，译有20余部图书。2008年进入出版业，从事技术图书编辑和审稿工作。

2007年创立知识分享网站“为之漫笔”（cn-cuckoo.com），翻译了大量国外经典技术文章。2012年下半年创立“A List Apart中文版”站点（alistapart.cn），旨在向中文读者译介这一国际顶级Web设计与开发杂志。他经常参加技术社区活动，曾在w3ctech 2012 Mobile上分享“Dive into Responsive Web Design”。2013年1月受邀在金山网络分享“响应式Web设计”，2013年3月应邀在奇虎360分享“JS的国”。

目前正在翻译Unix奠基人Brian Kernighan的新书*D is for Digital*。

TURING 图灵程序设计丛书

Stylin' with CSS
A Designer's Guide
Third Edition

CSS设计指南

（第3版）

[英] Charles Wyke-Smith 著
李松峰 译

人民邮电出版社
北 京

图书在版编目（C I P）数据

CSS设计指南 : 第3版 / (英) 史密斯 (Smith,C.W.)
著 ; 李松峰译. -- 北京 : 人民邮电出版社, 2013.5
(图灵程序设计丛书)
书名原文: Stylin’ with CSS:a designer’s guide
ISBN 978-7-115-31364-5

Ⅰ. ①C… Ⅱ. ①史… ②李… Ⅲ. ①网页制作工具－
指南 Ⅳ. ①TP393.092-62

中国版本图书馆CIP数据核字(2013)第062215号

内 容 提 要

本书是一本面向初中级读者的经典设计指南。全书共分 8 章，前 4 章分别介绍了 HTML 标记和文档结构、CSS 工作原理、定位元素、字体和文本，对规则、声明、层叠、特指度、选择符等基本概念进行了详细解读。随后 4 章介绍了页面布局、界面组件，CSS3 圆角、阴影、渐变、多背景等视觉设计技巧，最后还对如何实现最前沿的响应式设计进行了通俗易懂的演示。

本书适合 CSS 初中级读者阅读。

图灵程序设计丛书

CSS设计指南（第3版）

◆ 著　　[英] Charles Wyke-Smith
　译　　李松峰
　责任编辑　毛倩倩

◆ 人民邮电出版社出版发行　　北京市崇文区夕照寺街14号
　邮编　100061　　电子邮件　315@ptpress.com.cn
　网址　http://www.ptpress.com.cn
　北京　　　　　　印刷

◆ 开本：800×1000　1/16
　印张：19
　字数：440千字　　　　2013年 5 月第 1 版
　印数：1－4 000册　　　2013年 5 月北京第 1 次印刷

著作权合同登记号　图字：01-2013-2647号

ISBN 978-7-115-31364-5

定价：59.00元

读者服务热线：(010)51095186转604　印装质量热线：(010)67129223

反盗版热线：(010)67171154

版 权 声 明

两个“之最”（译者序）

或许你不敢相信，这本书创下了不少“之最”。虽然只能说是我自己经历过的“之最”，但说不定它们也是“中国之最”，甚至“世界之最”。

首先，这是翻译遍数最多的一本书。2006 年，我翻译了本书第 1 版。当时是为了学习翻译而翻译，并没有出版（也没有出版社引进），翻译稿至今仍静静地躺在电脑硬盘的“翻译档案”里。2008 年，图灵引进了本书第 2 版，于是我作为译者，第二次重新翻译了这本书。结果就是受到读者好评的《写给大家看的 CSS 书（第 2 版）》。2012 年，本书第 3 版面世，由于内容更新超过 80%，所以我又一次重新翻译了它。现在，这本书的全稿已经跟读者（至少是购买电子版的读者）见面了。但鲜为人知的是，我在近 7 年的时间内，翻译了这本书的三个版本。同一本书，三个版本，每一版都翻译一遍，这在我的翻译生涯里是绝无仅有的，是为“最多”。

其次，这是翻译模式最新的一本书。这本书采用了在线翻译、按章发布的模式。所谓按章发布，就是翻译一章、编辑一章、发布一章。对于购买了电子版的读者，这种模式不仅大大缩短了出版周期，而且使翻译过程充满互动性。传统纸质书从开始翻译到最终印刷上市，一般都需要 6 个月以上的时间，有的甚至更长。这么长的周期内，译者、编辑与读者“不见面”，翻译质量如何，内容有没有吸引力，读者全然不知。按章发布之后，对读者来说，可以更早地、同步地读到新书，掌握新知识；对出版方而言，则向前延长了图书的寿命。更重要的是，这为译者、编辑与读者的交流和互动提供了可能。这种模式对于我而言是前所未有的，是为“最新”。

按章发布是一种敏捷出版实践。敏捷软件开发讲究快速迭代、持续交付。套用在敏捷出版上，就是译、编、读每章都会有迭代，而全书内容以章为单位持续地交付给读者。具体来说，每一次迭代从交付每一章的内容开始。作为译者，我会在交付每一章的时候写一段简短的导读文字，概述一章的看点和精彩之处，通过社区消息系

统群发给所有读者。读者在阅读期间，会反馈勘误或者留下读书笔记。勘误信息可以立即得到纠正，从而减少将来印在纸质书中的错误。而读书笔记则有助于增进读者间的互动，加深读者对内容的理解。从读者反馈来看，这种模式较传统出版模式是一个非常大的进步，得到了读者的认可。比如，读者 cisolarix 就说：“及时对读者进行响应，很赞啊。比传统的出版方式好多了。”

从某种意义上讲，说这本书是参与翻译人数最多的一本也不为过。为此，我把写这篇译者序时给这本书留言的读者列出来，感谢他（她）们提出的勘误、对本书的肯定和敢于留下自己的“手迹”：

nikogu、cisolarix 、北秋、小杜、_ituring、m1a0、小翼、igoogler、Ivan Yan、newiuce、Jimmy 哥、流年、langford、karaschee、曹小坤、冰雅冷轩、acmerfight、Lemoncolaz、larrycai、level55_474、fejustin、球球饭团、zhangweifang、长弓、hidoos、rocflytosky、no_particular、xzhbao、hidoos、zhaoshuxiang、Pippola! 、ivanzhaowy、karaschee、lyuehh、陈陆扬

当然，这里边要特别感谢一位读者：流年。他在阅读过程中写下了很多心得体会，为其他读者提供了拓展性知识。他的留言让我反思：中国读者实际上不是不善于交流，而是对基础知识掌握不够牢固，不敢交流。就如同有人提到用英文写作时所说的，最难的不是怎么写，而是写什么。英文语法错误其实是别人容易纠正的，但你到底想要表达什么则是别人无法知道的。具体到 CSS，虽然不难学，但真正彻底把原理和基本概念全搞通也不是件容易的事。如果你想让自己真正掌握 CSS，这本书就是一个不错的起点。

2013 年 1 月 北京 名佳花园

前　　言

对网页设计师来说，这是一个激动人心的时刻！互联网已经囊括了几乎所有媒体，有线电视、CD 和 DVD 已经被 Hulu、Netflix、Pandora 和 Spotify 等这些按需取用的在线服务取代。

当然，访问这些媒体的设备同样异彩纷呈。有传统的台式电脑、笔记本电脑，还有平板电脑、智能手机，甚至 60 英寸的大屏幕电视。

而保障所有这些设备和媒体顺畅运作，有一套全新的技术标准：HTML5、CSS3 和 JavaScript。

五年前，在写作本书第 2 版的时候，曾经出现过一种死板又复杂的 HTML 标准——基于 XML 的 XHTML。由于 XHTML 难以适应千变万化和快速发展的互联网开发需求，苹果、谋智和欧朋联手组建了 Web Hypertext Application Technology Working Group（WHATWG，Web 超文本应用技术工作组）。这个组织的宗旨，就是让 W3C 为推广 XHTML 标准而抛弃的 HTML 4 起死回生。然后，我们就看到了涅盘重生的 HTML5。过去三年间，HTML5 迅速取代 XHTML，在 Web 设计与开发领域获得了广泛认同。

HTML5 是为今天的多媒体互联网而生的，因此它提供了一整套 API，支持视频、音频、图形、地理定位、数据存储等等。HTML5 还为结构化文档提供了很多新的元素（`section`、`article`、`nav`，等等）。而在此前，没有什么语义的 `div` 外加标识性的类名和 ID，曾经顶替过这些新元素的角色。但顶替实非长久之计，它们会限制标记本身的语义和可移植性。

在 HTML 标准从 HTML 4 到 XHTML 再向 HTML5 过渡的过程中，CSS3 则取得了越来越多浏览器的支持。作为网页美容工具箱，CSS3 吸纳了大量的建议。由于规模实

在太大，于是就分成了众多模块，分别由不同的团队负责每个模块的制定工作。

经过了长久的等待，我们终于可以使用 CSS3 的新功能了，比如说渐变、过渡、变换、阴影、圆角，等等。不过，不少使用旧版本浏览器的用户恐怕还是无法体验到这些新“特效”。对不完全支持 CSS3 的浏览器，可以使用 Modernizr 这个 JavaScript 文件来检测出它们到底支持或不支持哪项功能。对于不支持的功能，可以写一些后备代码（替代代码）或者使用腻子脚本（模拟 CSS3 功能的 JavaScript 代码）。细节就不在这儿说了，看本书附录吧。

今天的互联网不仅变得更加“亲民”，而且相对以往，开发人员的生活也轻松不少。就拿现在这本书来说吧，写作和编码花了我几百个小时的时间，无数个白天和黑夜，还有不计其数杯咖啡。但我的感觉就像是参加一个庆祝聚会。因为在示例和代码的背后，本书实际上代表了当今 Web 设计领域最前沿的成果，而这些成果曾经是很多人翘首企盼的美好未来。曾经的梦想如今已成为现实。因此，要感谢 Jeffrey Zeldman、Ian Hicks，还有很多我说不上名字来的人。正是他们的努力和坚持，才让曾经的 Web 标准化之梦终于成真。这种感觉，就像一个爬山的人一步一步艰难地向上走，而突然有一天发现自己已经爬到了山顶一样。想想看，这是一个多么大的突破呀：以前，即使要写一个最简单的布局都必须考虑旧版浏览器，为此我得花数页篇幅向读者解释那些“歪招儿”，但现在我既可以省下我的笔墨，也可以不再浪费了宝贵的纸张了；以前，要写出一个阴影或圆角效果来，我得机关算尽地组织复杂的图片和嵌套的 DIV，但现在只要一行 CSS 代码就可以搞定了；以前，辛辛苦苦写好一个页面，在不同浏览器里看起来往往非常不一样，但现在所有新版本浏览器都能完整一致地呈现它了。

所以，这个新版本是面向未来的。我没有像写本书的前两个版本那样，连篇累牍地跟大家讲怎么解决浏览器的不兼容问题，而是尽量聚焦于 HTML5、CSS3 以及现代浏览器所能实现的新功能。Internet Explorer 9、Firefox、Chrome、Safari，还有 Opera（对了，它会自动更新），这些浏览器的行为能够保持一致。而旧版本浏览器（尤其是 IE8 及更低版本）的用户与日俱减。至于怎么为这些过时的家伙提供服务，大家看附录。本书的主体内容讲的只是今天和未来的 CSS。

只需了解最基本的概念

要掌握 CSS，不需要你才华横溢，也不需要你是编程高手。如果你很有才华也有编程经验，那当然更好。本书只要求你对 HTML 和 CSS 有基本的概念，了解一些最佳实践。然后，你可以通过本书夯实自己的基础，加深自己对 CSS 的理解。CSS3 涉及的功能太多了，有一些本书连提都没有提到。不过没关系，相信你只要把本书读懂、吃透，自己学会使用那些新玩艺儿绝对不在话下。当然啦，读者通过自学丰富自己的知识和技能，也是本书作者（也包括译者）由衷期盼的。

下载代码，别自己敲

从本书网站上可以下载本书所有示例的代码，网址为：http://www.stylinwithcss.com。我建议大家下载，不光是因为下载简单省事儿，还因为我可能会在发现问题后更新这些代码。另外，也欢迎大家到我的网站上提交勘误和问题。这个网站有我的博客，欢迎光临，也许你能从中得到有用的信息，或者受到一些启发。当然，更希望大家看完文章后给我留言、写评论。

最后，感谢你把这本书买回来。希望这本书能真正帮到你，特别是能够帮你实现自己的 Web 之梦。

致　　谢

首先感谢 Peachpit 的出版团队。感谢 Micheal Nolan 鼓励我写这个版本并保证选题通过,感谢项目编辑 Nancy Peterson 的指导、见解和耐心,还要感谢出版人 Nance Ruenzel 在 7 年间连续出版我的书。

对于编辑团队，我要感谢对语法较真儿的校对编辑 Darren Meiss，感谢具有超强统筹能力的生产编辑 Katerina Malone，还要感谢 Karin Arrigoni 制作了完备的索引。

非常感谢我的老朋友、技术编辑 Curtis Blanton，感谢他对代码和解说的详尽反馈意见——的确太厉害了，Curtis！另外也感谢程序员 Jeffrey Johnson 和 Isaac Shapira 对本书的建议和支持。

最后，特别感谢我妻子 Beth Bast。她作为执行编辑和排版，无数遍通读和编辑了我的手稿，制作了本书的图形并在 InDesign 里排出了整本书。这项工作持续了 5 个月，她的努力和汗水体现在了本书的每一页里。谢谢你，亲爱的，没有你就不会有这本书。

我想用力抱一抱我的女儿 Jemma 和 Lucy，她们在父母合作编写这本书期间表现出了非凡的耐心。我们爱你们！

——Charles Wyke-Smith

2012 年 9 月 24 日于南卡罗来纳州查尔斯顿市

目　录

目　录

1 HTML 标记与文档结构

这是一本讲 CSS 的书，所以，第 1 章要先讲讲 HTML（Hypertext Markup Language，超文本标记语言）。

之所以从 HTML 讲起，是因为 CSS 的用途就是为 HTML 标记添加样式。所以，你首先得知道怎么编写和构造 HTML 标记，才能让 CSS 更方便地为它添加样式。每个网页在呱呱坠地时都只包含 HTML 标记,因为用 HTML 来标记内容是做任何网页的头一件事。所谓内容，就是你想要传达给读者的那些东西，比如文字、图片、音频和视频。

用 HTML 标记内容的目的是为了赋予网页语义（semantic）。换句话说，就是要给你的网页内容赋予某些用户代理（user agent）能够理解的含义。我们平常用的浏览器、给视障用户朗读网页的屏幕阅读器，以及搜索引擎放出的 Web 爬虫都是用户代理，它们需要显示、朗读和分析网页。HTML 规定了一组标签，用来给内容打上不同的标记。每个标签都是对它所包含内容的一种描述。最常用的 HTML 标签描述的是标题 、段落、链接和图片。目前，HTML 一共有 114 个标签，但按照 80/20 原则，使用其中 25 个左右的标签就可以满足 80%的标记需要。要想知道和了解 HTML 标签，可以看看本书网站：http://www.stylinwithcss.com。

在给内容都打上标记之后，就可以使用 CSS 来给标签添加样式了。添加样式的根据有标签名、标签属性（如 `id` 和 `class`），以及标签与其他标签在标记中的相对位置关系，等等。HTML 标签也会构成一个层次化的文档，从而可以通过 CSS 来设置网页的布局，为每个元素应用你想要的样式。

HTML 最新的版本 HTML5，又新规定了一批结构化标签，用于对相关内容的标签进行分组，从而更好地规范网页的整体结构。这些新标签包括<header>、<nav>（即 navigation，导航）、<article>、<section>、<aside>和<footer>。比如，对于一组可以让用户导航到网站中其他页面的链接，就可以使用<nav>标签把它们独立地标记出来。而在<article>里，则可以组织起来一篇既包含多个标题，又包含文本段落的博客文章。

在 HTML5 标准出现之前，创建页面的结构只能使用一些几乎没有什么语义的标签，比如<div>和<span>。而现在，我们有了一套专门的结构化标签。

为了进一步说明上述思想，接下来我们就看一看到底怎么给内容加标记，并展示怎么用 HTML 创建出一种标签层次，以便应用 CSS 和 JavaScript。

1.1　HTML 标记基础

对于每个包含内容的元素（比如标题、段落和图片），根据它所包含的内容是不是文本，有两种不同的方式给它们加标记，一种是使用闭合标签，另一种是使用自闭合标签。

1.1.1　文本用闭合标签

闭合标签的基本格式如下：

```
<标签名>文本内容</标签名>
```

可以给这个标签添加一些属性，比如：

```
<标签名 属性_1="属性值" 属性_2="属性值">文本内容</标签名>
```

标题、段落等文本元素都要求闭合标签，也就是要有一个开标签和一个闭标签，比如：

```
<h1>Words by Dogsworth</h1>
<p>I wandered lonely as a dog.</p>
```

可见，每个标签都由一对包含标签名的尖括号组成。标签名通常是一个（表示内容类型的）英文单词的简写形式或者其中的一个字母。闭标签与开标签的不同之处是标签名前面多一个斜杠。

位于开标签和闭标签之间的，就是将来会显示在浏览器中的内容；标签本身不会显示。

开标签和闭标签明确地标识出了标题和段落的起始位置。注意，前面的标题标签名中有一个数字 1，那是因为 HTML 有 6 级标题，<h1>表示最高级别的标题。

1.1.2　引用内容用自闭合标签

```
<标签名 属性_1="属性值" 属性_n="属性值" />
```

非文本内容是通过自闭合标签显示的。闭合标签与自闭合标签的区别在于，闭合标签包含的是会显示的实际内容，而自闭合标签只是给浏览器提供一个对要显示内容的引用。浏览器会在 HTML 页面加载的时候，额外向服务器发送请求，以取得自闭合标签引用的内容。

下面就是使用自闭合标签标记的一张图片。

```
<img src="images/cisco.jpg" alt="My dog Cisco">
```

关闭标签

XHTML 标准严格规定所有标签都必须关闭，但 HTML5 标准在这方面则要宽松得多。换句话说，在 HTML5 网页里，某些闭标签是可以省略不写的。要了解 HTML5 语法，可以参考这个链接 http://dev.w3.org/html5/html-author/#syntactic-overview。

即使你在标记段落时只写了一个段落的开标签，没有写与之对应的闭标签，仍然可以通过 HTML5 标准验证。不过，我个人习惯于关闭所有标签，因为这样可以更清楚地知道自己标记的结构，同时也能保证不让自己忘了关闭那些应该关闭的标签。我建议读者也这样做，本书中的代码示例也会关闭所有标签。

对于自闭合标签，XHTML 要求必须这样写：

```
<img src="images/cisco.jpg" alt="My dog Cisco" />
```

而在 HTML5 中，可以省略最后那个表示关闭的斜杠，写成：

```
<img src="images/cisco.jpg" alt="My dog Cisco" >
```

不过，我还是习惯于给自闭合标签的末尾也加上一个空格和斜杠。HTML5 会忽略这个斜杠，而我在检查代码结构时，通过它一眼就能知道这个标签已经关闭了，所以它不包含后面的标签。

1.1.3　属性

属性负责为浏览器提供有关标签的额外信息。比如说，前面例子中的<img>标签有两个属性。一个属性是 src（source，来源），属性值为 cisco.jpg。这个属性定义了图片的来源是一个名为 cisco.jpg 的文件。另一个属性是 alt（alternative，替代内容），定义的是在图片因故未能加载成功时在屏幕上显示的文本。



视障用户使用的屏幕阅读器会大声读出 alt 属性的内容，因此一定要给<img>标签的这个 alt 属性添加让人一听（或一看）就能明白的内容。

每个 HTML 标签都可以添加属性。在稍后的例子中我们会看到，class 和 id 等属性，几乎可以适用于任何标签。而另一些属性，比如 src，则只适用于类似<img>这样的引用源文件的标签。



要想了解所有 HTML 标签和属性，可以参考 HTML Dog 网站：http://htmldog.com/reference/htmltags。

本章使用了哪些 HTML 标签

以下是本章使用的块级（block）和行内（inline）标签。至于什么是块级标签，什么是行内标签，本章稍后再作介绍。

块级标签

- <h1>–<h6>：6 级标签，<h1>表示最重要
- <p>：段落
- ：有序列表
- ：列表项
- <blockquote>：独立引用

行内标签

- <a>：链接（anchor，锚）
- <img>：图片
- <em>：斜体
- <strong>：重要
- <abbr>：简写

- <cite>：引证
- <q>：文本内引用

1.1.4 标题与段落

到现在为止，我们用得最多的还是标题和段落标签。一般来说，网页都会以一个<h1>标签开头，其中的文本用于告诉读者这个网页是干什么的。然后用<h2>标记下一级内容，或许是一个副标题，然后才是<h3>，依此类推。

<h1>不仅是最大最突出的标题（除非你用 CSS 缩小它的字号），搜索引擎也会将其视为仅次于<title>标签的另一个搜索关键词的来源。

段落用于标记主要的文本内容，是所有文本元素中出场率最高的一个。简言之，只要有不适合放在其他文本标签中的文本，都可以把它放一个段落里面。

接下来，我们会谈一谈文档流（document flow）、行内元素和块级元素的概念，介绍一下每个元素在页面中怎么形成一个方盒子（box）。HTML 中的这些基本概念，对于使用 CSS 快速有效地为文档添加样式，使其变成你期望的布局和外观具有重大意义。

1.1.5 复合元素

HTML 不仅规定了标题、图片和段落等基本的内容标记，还规定了用于创建列表、表格和表单等复杂用户界面组件的标记，这些就是所谓的复合元素，即它们是由多个标签共同构成的。

比如，<li>是一个列表项，它只在<ol>（ordered list，有序列表）和<ul>（unordered list，无序列表）这两种列表标签中才有效，在<dl>（definition list，定义列表）中则无效。下面是一个包含三个列表项<li>的有序列表<ol>的例子：

```
<ol>
    <li>Save HTML file</li>
    <li>Move file to Web server via FTP</li>
    <li>Preview in browser</li>
</ol>
```

图 1-1 展示了这个列表在浏览器中呈现的效果。

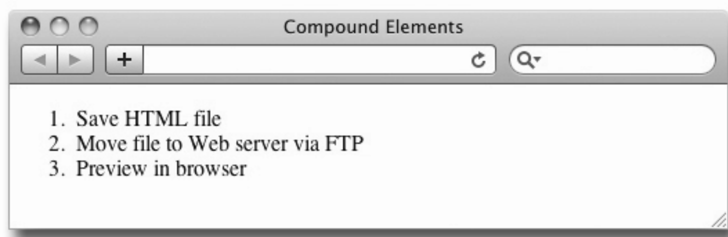


图 1–1　有序列表示例。浏览器会自动给每个列表项编号

这个例子告诉了我们两个基本知识点。首先，某些标签（如这里的<ol>）要求其他标签（即这里的<li>）与之共同出现。其次，我们说这里的列表项<li>“被嵌套”在有序列表<ol>中，因为有序列表的开标签位于所有列表项之前，闭标签位于所有列表项之后。标签嵌套是我们必须要理解的一个至关重要的概念。

1.1.6　嵌套标签

在上面的例子中，基于<li>标签与<ol>标签的嵌套关系，我们还可以说<li>标签是<ol>标签的子标签（或子元素），或者说<ol>标签是<li>标签的父标签（父元素）。

下面是使用<em>（emphasis，强调）标签来强调段落中一个单词的例子。遗憾的是，其中的<em>子标签嵌套在<p>标签里的方式是错误的。

```
<p>That car is <em>fast</p>.</em>
```

正确的写法应该是下面这样的。

```
<p>That car is <em>fast</em>.</p>
```

请记住，要在一个标签里嵌套另一个标签（也就是前者的开标签写在后者的开标签之前），必须要先关闭后一个标签再关闭前面那个标签。上面第一个例子搞错了，第二个例子没有问题。

HTML 文档的结构正是通过类似这样的标签嵌套，以及就此建立起来标签间的“父－子”关系形成的。下面，我们就通过分析一个 HTML 文档的结构，来实际地演练一番。

1.2 HTML 文档剖析

有几个 HTML 元素是所有 HTML 文档（也就是网页）中都必须包含的。这些元素为内容提供了框架，有了这个框架才能正确显示内容。可以把这些必要的元素想象成一个模板页面里必须得有的元素。很多网页编写工具，比如 Adobe Dreamweaver，会在每次创建新 HTML 页面时自动给你生成一个包含这些元素的模板页面。



HTML5 标准推出之后，HTML 文档的结构一下子得到了极大简化。就算你的 Web 开发经验不多，估计都会对之前 HTML 和 XHTML 复杂的文档类型（doctype），包括过渡型和严格型，以及那些元数据标签（<meta>）记忆犹新。所有这些，在 HTML5 中都已经被更简单的语法取代，而且虽然语法简单但却能够与之前的 HTML 版本兼容。

1.2.1 HTML模板

今天，按照 HTML5 语法编写的最简单的 HTML 页面的模板可以写成这样：

```
<!DOCTYPE html>
<html>
    <head>
    <meta charset="utf-8" />
    <title>An HTML Template</title>
    </head>
    <body>
        <!-- 这里是网页内容 -->
    </body>
</html>
```

模板的第一行代码是一种新语法，或者说是一种简化的 DOCTYPE，这一行就是为了声明：“以下是一个 HTML 文档。”注意，这个标签不用关闭。

然后是<html>标签，也就是所谓的根级标签，因为页面中所有的其他标签都嵌套在这个标签内部，而且它的闭标签也是整个页面中的最后一个闭标签。<html>标签只有两个直接的子标签：<head>和<body>。



使用 HTML 注释标签，可以为你自己或者其他将来可能会修改这个页面的人写一些备注。HTML 注释以<!--开头，以-->结尾，注释内容就写在它们中间。浏览器在加载页面时，会忽略注释，不会显示其中的内容。

帮助浏览器理解页面的信息都包含在<head>标签中。在这个最简单的例子里，<head>标签里只包含<meta>和<title>两个标签。其中，<meta>标签有一个 charset 属性，它是在告诉浏览器这个页面使用的是 UTF-8 编码。而<title> 标签中的文本会在页面显示时，作为整个页面的标题出现在浏览器窗口顶部的标题栏中。上面模板中页面的标题是“An HTML Template”。

关于<title>标签

搜索引擎会给<title>标签中的文字内容赋予很高的权重，而且这些文字也会作为网页标题出现在搜索结果列表中。为此，千万不要让那些“欢迎光临我的网页！”之类的废话占据你的<title>标签。一定要让这些文字简洁明确，而且包含目标读者在搜索你的网页内容时会使用的关键词。

而<body>标签则包含着标记所有内容的 HTML 元素。接下来，我们先把原来的注释替换成另外两个标签，然后再看看页面在浏览器中显示的效果。

在上面模板的<body>标签中，我们再添加两个标签：

```
<body>
    <h1>Stylin' with CSS</h1>
    <p>Great Web pages start with great HTML markup!</p>
</body>
```

图 1-2 展示了当前页面在浏览器中显示的效果。



图 1–2　带有一个标题和一个段落的示例模板页面

浏览器把包含内容的元素在页面中自上而下地一一排列，起点是页面的左上角。仔细看一看，会发现页面中标题和段落文本的字号不一样，但字体都是 Times Roman，而且标题与段落之间还有一定的间距。很显然，浏览器已经为页面应用了一些基本的样式。这些初始（默认）的样式，来自浏览器内置的一个 CSS 样式表，它只会为每个 HTML 元素添加一些最基本，当然也谈不上美观的样式。

接下来再给模板添加两个比较常用的 HTML 元素，一个链接，一个图片。

```
<body>
    <h1>Stylin' with CSS</h1>
    <p>Great Web pages start with great HTML markup!</p>
    <a href="http://www.stylinwithcss.com">My Books</a>
    <img src="images/cisco.jpg" alt="My dog Cisco" />
</body>
```

图 1-3 展示了页面现在的效果。

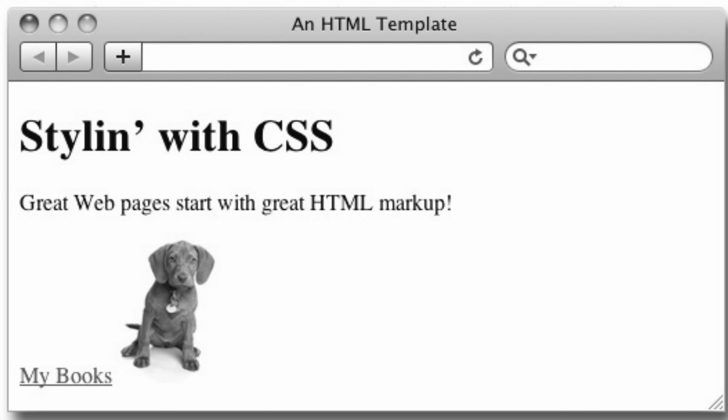


图 1–3　页面中目前包含着标题、段落、链接和图片

链接是使用<a>标签创建的，该标签有一个必需的属性 href（hyperlink reference，超链接引用），其中包含着链接指向的页面的 URL。

前面我们也提到过<img>标签，而现在你终于看到它的效果了。在这个标签里，同样是给出图片所在位置的 URL，要使用 src（source，来源）而不是 href 属性。

有一点你可能注意到了：虽然标题和段落是上下堆叠在一起的，但链接和图片却是并排显示的。为什么会这样呢？因为标题和段落是块级元素，而链接和图片是行内元素。

1.2.2　块级元素和行内元素

图 1-3 展示了所谓“文档流”的效果，也就是 HTML 元素会按照它们各自在标记中出现的先后顺序，依次从页面上方“流向”下方。浏览器的样式表之所以提供这种流动式效果，就是为了让那些简单但却正确标记了 HTML 的内容，能够以朴素但却可用的方式显示出来。CSS 的妙处就在于把这种实用主义的默认显示效果，转换成既有吸引力又直观的页面设计。

几乎所有 HTML 元素的 `display` 属性值要么为 `block`，要么为 `inline`。最明显的一个例外是 `table` 元素，它有自己特殊的 `display` 属性值。

块级元素（比如标题和段落）会相互堆叠在一起沿页面向下排列，每个元素分别占一行。而行内元素（比如链接和图片）则会相互并列，只有在空间不足以并列的情况下才会折到下一行显示。

无论你想了解哪个 HTML 元素，第一个要问的问题都应该是：它是块级元素，还是行内元素？知道了这一点之后，就可以在编写标记的时候，预想到某个元素在初始状态下是如何定位的，这样才能进一步想好将来怎么用 CSS 重新定位它。

使用块级元素和行内元素构建页面

这里是一个完全由标题和段落组成的页面。为了保证屏幕截图和代码清晰，我故意把段落都弄得非常短。页面的标记如下。

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="utf-8" />
    <title>Block and Inline Elements</title>
</head>

<body>
    <h1>Types of Guitars</h1>
    <p>Guitars come in two main types: electric and acoustic.</p>
    <h2>Acoustic Guitars</h2>
    <p>Acoustic guitars have a large hollow body that projects the sound of the
        strings.</p>
```

```
    <h3>Nylon String Acoustic Guitars</h3>
    <p>Descendants of the gut-strung instruments of yore, nylon string guitars
       have a mellow tone.</p>
    <h3>Steel String Acoustic Guitars</h3>
    <p>Steel string guitars first appeared in country music and today most acoustic
       guitars have steel strings.</p>
    <h2>Electric Guitars</h2>
    <p>Electric guitars have a solid or hollow body with pickups that capture
       the string vibration so it can be amplified.</p>
    <h3>Solid Body Electric Guitars</h3>
    <p>Solid body electric guitars are commonly used in rock and country music.</p>
    <h3>Hollow Body Electric Guitars</h3>
    <p>Hollow body acoustic guitars are commonly used in blues and jazz.</p>
       </body>

</html>
```

整个页面的内容只有标题和段落。为清晰起见，这里省略了本该有的<article>等标签。

图 1-4 展示了以上代码在浏览器中显示的效果。

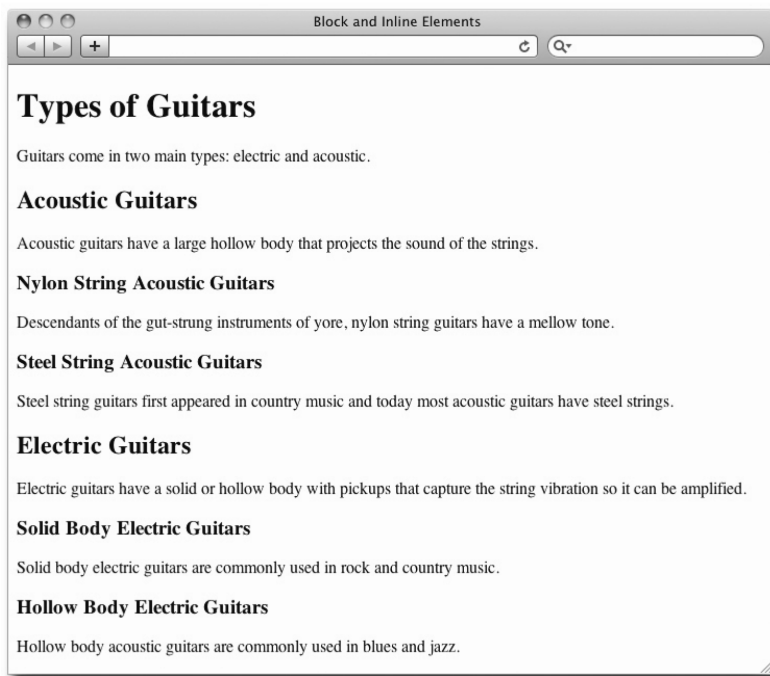


图 1-4　由标题和段落构成的页面

在以上代码和屏幕截图中，可以看到三种级别的标题。浏览器为每一级标题分别应用了不同的字号，从而让页面内容的层次显得非常分明。由于标题和段落都是块级元素，所以每个元素各占一行。

还有，页面四周都添加了一定的边距，所以文本才不会碰到浏览器窗口。而且，段落中的行与行之间也有距离。好了，边距的话题回头我们再仔细聊，还是先给页面添几张图片吧。如图 1-5 所示，当然你已经知道了，图片是行内元素。

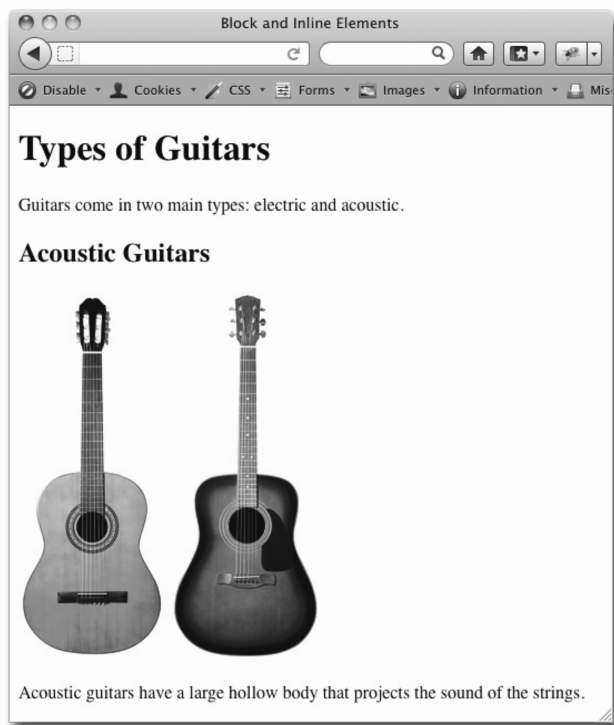


图 1-5　为页面添加两个行内元素（吉他图片）



标记中的两个<img>标签虽然分别位于两行，但这并不影响图片在浏览器中显示时的效果。图片是行内元素，所以它们显示的时候就会并列出现在一行上；而且，标签之间的空白（包括制表、回车和空格）都会被浏览器忽略[①]。因此，为了让代码的版式布局一目了然，你可以随意在标签之间添加换行和空格。一般推荐的做法都是子标签相对于父标签要有一个缩进。

① 根据 CSS 标准，默认情况下，多个空白会被浏览器叠加为一个。——译者注

以下是图 1-5 中展示的那部分页面对应的代码，其中包含两张图片。图片是行内元素，所以两把吉它会并排显示。

```
<body>
    <h1>Types of Guitars</h1>
    <p>Guitars come in two main types: electric and acoustic</p>
    <h2>Acoustic Guitars</h2>
    <img src="images/acoustic_nylon.jpg" alt="nylon string acoustic guitar" />
    <img src="images/acoustic_steel.jpg" alt="steel string acoustic guitar" />
    <p>Acoustic guitars have a large hollow body that projects the sound of the
        strings.</p>
</body>
```

现在，为了进一步加深对块级元素和行内元素的理解，我们在这个页面上搞一次小试验。这次试验要用到我最喜欢的一个开发工具，那就是 Web Developer 扩展。它是 Firefox 的一个扩展，安装之后它会生成一个菜单，里面包括很多查看 HTML、CSS 和 JavaScript 的工具。



要安装 Web Developer，在 Firefox 工具栏中选择“附加组件”，然后在“附加组件管理器”中搜索“Web Developer”。找到之后，点击安装。

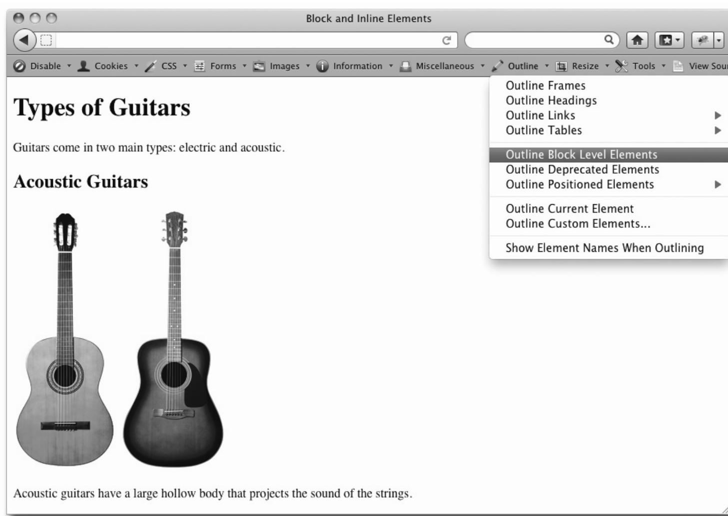


图 1-6　我在 Web Developer 工具条中选择了 Outline 菜单

在图 1-6 中，可以看到浏览器窗口顶部偏下的 Web Developer 工具条。这个工具条有很多功能，其中一个就是能让你更清晰地确定每个元素的位置，以及元素之间的嵌

套关系。如图所示，我选择了 Outline（轮廓）菜单中的 Outline Block Level Elements（显示块级元素的轮廓）命令，这个命令可以显示页面中块级元素方盒子的轮廓。

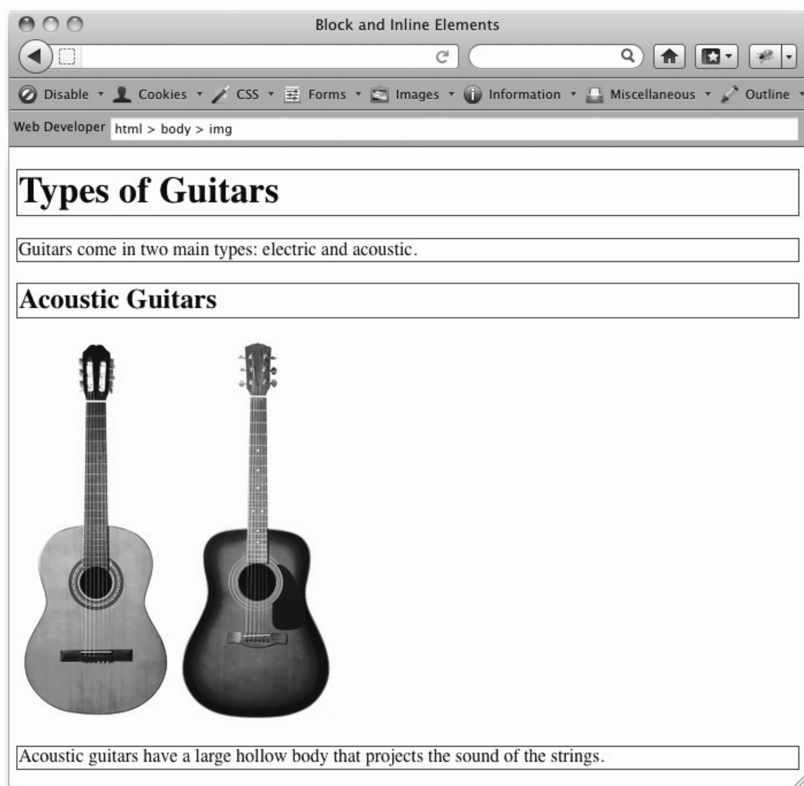


图 1-7　选择 Outline Block Level Elements 命令后，会显示块级元素盒子的实际大小以及相互之间的间距。没有显示行内元素的轮廓

显示块级元素的轮廓之后，通过图 1-7 就可以发现元素盒子比它们包含的文本要大一些。每个盒子的高度比内容稍微高一点，而宽度跟浏览器窗口一样宽！

块级元素盒子会扩展到与父元素同宽

在我们这个页面中，所有块级元素的父元素都是 body，而它的宽度默认与浏览器窗口一样宽（当然有少量边距）。因此，所有块级元素就与浏览器窗口一样宽了。说到这，相信你就能理解为什么块级元素始终会占一行了。对了，就是因为它们始终会保持与浏览器窗口同宽。这样一来，一个块级元素旁边也就没有空间容纳另一个块级元素了。

通过 Web Developer 工具条不能像显示块级元素轮廓那样，同时显示所有行内元素的轮廓。不过，通过 Outline 菜单中的 Outline Custom Elements（显示自定义元素的轮廓）命令可以显示这些元素的轮廓。行内元素盒子的行为与块级元素盒子的行为正好相反。

行内元素盒子会“收缩包裹”其内容，并且会尽可能包紧

说到这，你就可以理解为什么几个行内元素会并排显示在一行，而每个块级元素都会另起一行了。

1.2.3　嵌套的元素

现在，该看一看在标记中嵌套的 HTML 元素到了屏幕上是什么效果啦。前面例子中的所有元素都有一个共同的父元素——body。因此，Web Developer 通常不显示它。但通过在 Outline 菜单中选择 Outline Custom Elements 命令，可以自定义显示 body，如图 1-8 所示。

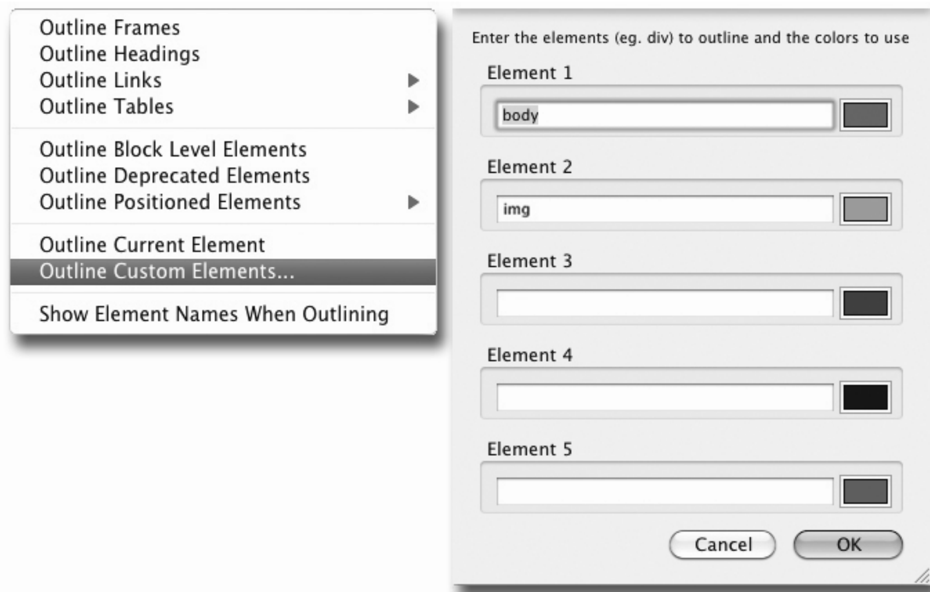


图 1-8　选择 Outline Custom Elements 命令后，在弹出的窗口中可以指定显示哪个元素以及用什么颜色显示

如图 1-9 所示，父元素 body 的盒子（最外面的轮廓），直接嵌套（也就是包裹）着所有子元素盒子（内部的轮廓）。其中，块级子元素扩展到与 body 元素同宽，后者默认与浏览器窗口同宽（但有一点外边距）。

在标记中嵌套的是 HTML 标签，而在屏幕上嵌套的则是一个个的盒子

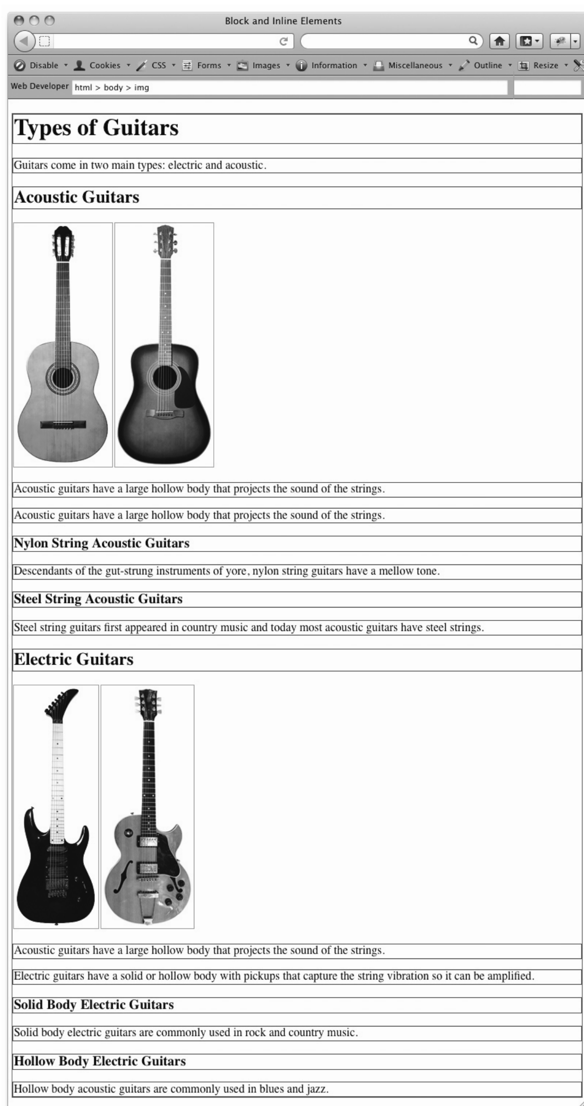


图 1–9　父元素 body 包围着它的子元素

在一个包含很多元素的页面中，盒子套盒子会越套越深，此时合理的 HTML 布局有助于通过标记看清页面结构，从而保证标签间正确的嵌套关系。我们推荐 HTML 标签每个层次相对于上一层次都缩进 4 个空格，如下面代码中的点所示。



在 Dreamweaver 等 HTML 编辑器中，每次按 Tab 键时就会缩进 4 个空格，从而保证缩进的一致性。

```
<nav id="toc">
....<ol>
........<li><a href="#">Introduction</a></li>
........<li><a href="#">Chapter 1</a></li>
........<li><a href="#">Chapter 2</a></li>
........<li><a href="#">Chapter 3</a></li>
....</ol>
</nav> <!-- end table of contents -->
```

两个在标记中嵌套的例子

我们再来看一个嵌套的例子，这一次使用 blockquote 元素。顾名思义，blockqoute 元素包含引用内容，而且在页面上看起来是独立的元素。请注意代码中使用的表示弯引号的 HTML 实例。

```
<blockquote>&ldquo;Sometimes you want to give up the guitar, you'll hate the
    guitar. But if you stick with it, you're gonna be rewarded.&rdquo;
    <cite>Jimi Hendrix</cite>  //使用 cite 元素包含作者姓名
</blockquote>
```

图 1-10 展示了这段代码在页面中显示的效果，为两个元素都加了轮廓。

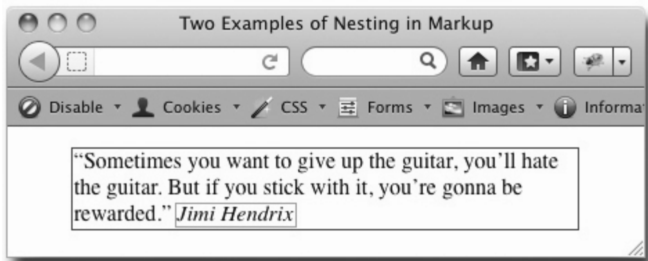


图 1-10　blockquote 元素默认会缩进

HTML 实体

HTML 实体常用于生成那些键盘上没有的印刷字符，比如 ™、† 、©，等等。HTML 实体以一个和号（&）开头，一个分号（;）结尾，二者之间是表示实体的字符串。在前面的例子中，两个实体的名字分别是“left-double-quote”（左双引号）和“right-double-quote”（右双引号）的简写。

Peachpit 另一位作者 Elizabeth Castro（她的书非常赞，在此强烈向大家推荐）在自己的一个网页中列出了常用的 HTML 实体：http://www.elizabethcastro.com/ html/extras/entities.html。

要注意的是，由于和号表示实体开头，所以在要显示和号的时候不能直接写和号，而要在 HTML 标签包含的文本中使用`&`实体，这样才能显示出&来。比如，写成 `Johnson & Johnson`，才会显示成：Johnson & Johnson。

显然，`<blockquote>` 标签也是一个块级元素（它名字里就有 block）。因为它的目的就是在页面中独立显示一块内容，所以作为块级元素非常合适。

嵌套在 `blockquote` 元素中位于引用文本之后的是 `cite` 元素，它是一个行内元素。因为文本结束后还有空间，所以它就显示在了段落最后一行上。从图 1-10 中也可以看到，`<cite>`标签的内容默认是斜体。

同样是在这个例子中，还可以看到两个 HTML 实体，`“`和`”`，分别用于生成能够正确打印出来的左、右双引号。使用这两个引号的实体，而不是按键盘上的 Shift-'（Shift-引号键），能让页面显示更加专业。

第二个嵌套的例子如下，是一个块级元素包含三个行内元素。

```
<p>It is <strong>absolutely critical</strong> that <em>everyone</em> does this
    <abbr title="as soon as possible">ASAP</abbr>!</p>
```

图 1-11 展示了 Firefox 浏览器中显示的效果。



图 1-11　一个段落中嵌套着三个分别表示重要性、强调和简写的标签

图 1-12 与图 1-11 一样，只不过显示了元素盒子的轮廓。



图 1-12　三个行内元素嵌套在一个块级元素中

这个例子除了能让你感到情况紧急之外，还能让你有如下收获。

- 文本被标记为段落，而其中包含三个行内元素。
- <strong>标签表示重要，默认以粗体显示。
- <em>标签表示强调，默认以斜体显示。
- <abbr>标签表示简写，在 Firefox 中默认会加上点下划线。

好了，我们已经知道了 HTML 标记怎么在页面中创建盒子了，也知道了嵌套标记实际上就是嵌套盒子。那么，接下来我们就可以聊一聊文档对象模型（Document Object Model，DOM）了。

1.3　文档对象模型

在开始学习 CSS 之前，关于 HTML 我们要介绍的最后一个知识点，就是 HTML 结构所对应的文档对象模型（以下简称“DOM”）。DOM 是从浏览器的视角来观察页面中的元素以及每个元素的属性，由此得出这些元素的一个家族树。通过 DOM，可以确定元素之间的相互关系。在 CSS 中引用 DOM 中特定的位置，就可以选中相应的 HTML 元素，并修改其样式属性。

我们就通过下面这个例子来理解 DOM 吧。

```
// HTML 代码正确缩进，以表明标签的层次关系
<body>
    <section>
        <h1>The Document Object Model</h1>
        <p>The page's HTML markup structure defines the DOM.</p>
```

```
    </section>
</body>
```

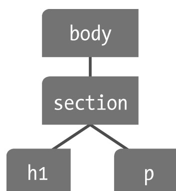


图 1-13　与标记对应的简单 DOM 结构图

上面的代码示例和图 1-13 展示了一个典型的页面结构，即结构化的标签（即<section>）包含着几个子内容标签（这里的<h1>和<p>）。代码中用缩进表示元素间的层次关系。这种层次关系也可以用垂直的、类似家族树的结构图表示。

图 1-14 展示了标明元素轮廓的 HTML 元素。

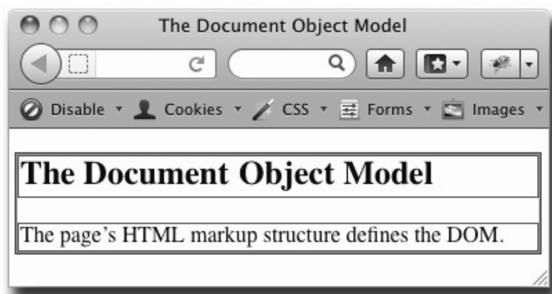


图 1-14　从图中可以看出，body 元素包含着 section 元素，后者进而包含一个标题和一个段落

对于这个例子中的 DOM 层次，我们可以作如下表述。

- section 是 h1 和 p 的父元素，也是直接祖先元素。
- h1 和 p 是 section 的子元素，也是直接后代元素。
- h1 和 p 是同胞元素，它们有共同的父元素 section。
- section、h1 和 p 是 body 的后代元素，或者下面的元素（嵌套在后者的内部）。
- section 和 body 是 h1 和 p 的祖先元素，或者上面的元素（在某一层次上包含后者）。

在本书后面几章里，我们会经常使用子元素、父元素、同胞元素、祖先元素和后代元素这些概念。所以，请读者务必先对 HTML 创建的 DOM 的层次有一个清晰的认识，搞明白这些概念的含义，以及它们之间的嵌套关系。

CSS 操作 DOM 的过程是先选择一个或一组元素，然后再修改这些元素的属性。通过 CSS 修改了元素后，比如修改了宽度或者在标记里插入一个伪元素，这些变化会立即在 DOM 中发生，并体现在页面上。

简言之，就是通过 HTML 标记来构建 DOM，然后在页面初次加载和用户与页面交互时，使用 CSS 来修改 DOM。

1.4　小结

本章讲解了 HTML 标签怎么为内容提供结构，以及每个元素会在屏幕上生成什么样的盒子。演示了块级元素和行内元素的区别，以及在元素互相嵌套的情况下，它们之间会存在的一种层次关系。你知道了标记中嵌套的 HTML 元素会生成屏幕上嵌套的盒子。最后，我们学习了 DOM，它是浏览器中文档的模型，而 CSS 可以修改 DOM 中元素的样式属性，从而修改页面本身的布局和外观。本章学习的这些基础知识，对于使用 CSS 为 HTML 添加样式至关重要。第 2 章，我们会从 CSS 的规则和机制入手，讲一讲它是怎么对 HTML 发生作用的。

2

CSS 工作原理

第 1 章我们了解了怎么通过 HTML 来创建文档结构。本章，我们来说一说 CSS 规则怎么为 HTML 添加样式，并解释层叠的工作机制——当元素的同一个样式属性有多种样式值的时候，CSS 就要靠层叠机制来决定最终应用哪种样式。

每个 HTML 元素都有一组样式属性，可以通过 CSS 来设定。这些属性涉及元素在屏幕上显示时的不同方面，比如在屏幕上位置、边框的宽度，文本内容的字体、字号和颜色，等等。CSS 就是一种先选择 HTML 元素，然后设定选中元素 CSS 属性的机制。CSS 选择符和要应用的样式构成了一条 CSS 规则。

好，我们就先聊一聊 CSS 规则，以及怎么把 CSS 规则应用到 HTML 元素。请读者跟着我做几个例子，这些例子都要用到下面这个简单的 HTML5 模板。这个模板的代码也可以登录 http://www.stylinwithcss.com 下载。



这段代码也展示了 CSS 注释与 HTML 注释在格式上的区别。

```
<!DOCTYPE html>
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html;
    charset=UTF-8" />
    <title>HTML5 Template</title>
    <style>
    /* CSS 规则放在<style>标签中 */
    </style>
```

```
</head>
<body>
    <!-- HTML 元素放在<body>标签中 -->
</body>
</html>
```

代码里包含一个 HTML 的<style>标签。通过这个标签可以把 CSS 样式直接写在文档中（用专业说法，是把 CSS 样式嵌入到文档中）。浏览器会负责把<style>标签中的 CSS 样式应用给<body>标签中的 HTML 元素（甚至包括<html>标签）。

2.1　剖析 CSS 规则

规则实际上就是一条完整的 CSS 指令。规则声明了要修改的元素和要应用给该元素的样式。

下面就是一条 CSS 规则，它可以把段落的文本设置为红色。

```
p {color:red;}
```

把它应用给以下 HTML 标记

```
<p>This text is very important!</p>
```

这个段落的文本就会显示为红色，如图 2-1 所示。

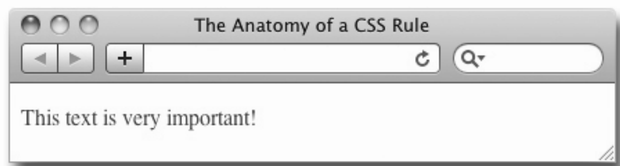


图 2-1　简单的 CSS 标签选择符给 HTML 段落文本添加颜色

把以上代码放在我们的 HTML5 模板中，就是这样。



这个 HTML 模板以及本书所有代码示例，都可以到这里下载：http://www. stylin-withcss.com。

```
<!DOCTYPE html>
<html>
```

```
<head>
    <meta charset="utf-8" />
    <title>HTML5 Template</title>
    <style>
        /*CSS 样式要嵌入在页面 head 元素中的<style>标签里*/
        p {color:red;}
    </style>
</head>
<body>
    <!-- HTML 元素放在<body>标签中 -->
    <p>This text is very important!</p>
</body>
</html>
```

为文档添加样式的三种方法

有三种方法可以把 CSS 添加到网页中，分别是写在元素标签里（也叫行内样式）、写在<style>标签里（也叫嵌入样式）和写在单独的 CSS 样式表中（也叫链接样式）。

行内样式

行内样式是写在特定 HTML 标签的 style 属性里的，比如：

```
<p>This paragraph simply takes on the browser's default paragraph style.</p>
<p style="font-size: 12px; font-weight:bold; font-style:italic; color:red;">By
adding inline CSS styling to this paragraph, you override the default styles.</p>
```

行内样式的作用范围非常有限。行内样式只能影响它所在的标签，而且总会覆盖嵌入样式和链接样式。

嵌入样式

嵌入的 CSS 样式是放在 HTML 文档的 head 元素中的，比如：

```
<head>
    <!-- 其他 head 元素（如 meta、title）放在这里 -->
    <style type="text/css">
        h1 {font-size:16px;}
        p {color:blue;}
    </style>
</head>
```

嵌入样式的应用范围仅限于当前页面。页面样式会覆盖外部样式表中的样式，但会被行内样式覆盖。像本书前面例子中那样使用嵌入方式为某个组件（比如菜单）设计样式是很方便的，因为 HTML 和 CSS 同在一页，可以互相参照。但是，等到 CSS 样式设计完毕，组件功能齐备之后，还是应该把相应的样式转移到外部样式表，以便其他页面也能共用相同的样式。

链接样式

在创建包含多个页面的网站时，需要把样式集中在一个单独的文件里，这个文件就叫样式表。样式表其实就是一个扩展名为.css 的文本文件。可以在任意多个 HTML 页面中链接同一个样式表文件，每个页面中只需加入类似下面的这一行代码即可：

```
<link href="styles.css" rel="stylesheet" type="text/css" />
```

链接样式的作用范围可以是整个网站。只要使用<link>标签把样式表链接到每个页面，相应的页面就可以使用其中的样式。随后，只要修改了样式表中的样式，改动就会在所有被选中的元素上体现出来，无论这个元素在哪个页面里。这样，既可以做到全站页面外观统一，又便于整站样式更新。

除了以上三种为页面添加样式的方法，还有一种在样式表中链接其他样式表的方法，那就应用 @import 指令（是一种 at 规则）：

```
@import url(css/styles2.css)
```

要注意的是，@import 指令必须出现在样式表中其他样式之前，否则@import 引用的样式表不会被加载。

有一点很重要，那就是 CSS 样式是通过<style>标签嵌入到页面里的。当浏览器遇到开标签<style>时，就会由解释 HTML 代码切换为解释 CSS 代码。等遇到闭标签</style>时，它会再切换回解释 HTML 代码。



对于写在样式表里的样式，就不需要<style>标签了。如果你在样式表里加上这个标签，样式表中的样式就不会被浏览器加载了。

本章后面的例子都将使用这个模板。读者可以把这个模板写到一个文本文件里，然后保存为扩展名是.html 的文件。之后，只要像前面那样，把每个例子的 CSS 和 HTML 代码复制粘贴到这个模板中就行了。粘贴后保存，再在浏览器中打开，就可以看到效果。

CSS规则命名惯例

CSS 规则由选择符和声明两部分组成，其中选择符用于指出规则所要选择的元素（图 2-2 的示例选择符是要选择段落），声明则又由两部分组成：属性和值。属性指出要影响元素哪方面的样式（图 2-2 的示例属性影响的是文本颜色），值就是属性的一个新状态（图 2-2 中是红色）。

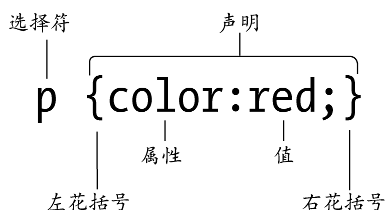


图 2–2　CSS 规则分两部分，即选择符和声明。声明又由两部分组成，即属性和值。声明包含在一对花括号内

总体来看，选择符后面是左花括号，然后是以冒号分隔的属性和值，二者构成声明。每个声明以分号结尾，最后是结束规则的右花括号。

建议读者仔细研究一下图 2-2，搞明白这张图，前面所有这些术语就都明白了。后面我们经常会提到这些术语。

对这个基本的结构，有三种方法可以进行扩展。

第一种方法：多个声明包含在一条规则里。

```
p {color:red; font-size:12px; font-weight:bold;}
```

这样，一条规则就可以把段落文本设置成红色，12 像素大，粗体。

注意三个声明末尾都有一个分号，以示分隔。其中，最后一个位于右花括号之前的分号是可选的，但我始终都会加上它。因为将来再添加声明时就不用再记着加了。

有读者可能会好奇地想，这些属性（比如 `font-size` 和 `color`）还有其他什么值呢？比如，指定颜色值的时候是不是可以不用颜色名，而用 RGB（red,green,blue）格式？当然，可以。不过，我想还是容我把选择符的工作机制讲完吧。本章后面，我们再详细讲规则的声明部分。

第二种方法：多个选择符组合在一起。如果想让<h1>、<h2>和<h3>的文本都变成蓝色，粗体，可以这样分别写：

```
h1 {color:blue; font-weight:bold;}
h2 {color:blue; font-weight:bold;}
h3 {color:blue; font-weight:bold;}
```

但其实，把三个选择符组合在一起也可以，这样就能减少重复输入：

```
h1, h2, h3 {color:blue; font-weight:bold;}
```

千万注意每个选择符之间要用逗号分隔（最后一个后面不用加）。至于选择符之间的空格，随意，可加可不加，但加了好像看得更清楚。

第三种方法：多条规则应用给一个选择符。 假设，你在写完前面那条规则后，又想只把 h3 变成斜体，那可以再为 h3 写一条规则：

```
h1, h2, h3 {color:blue; font-weight:bold;}
h3 {font-style:italic;}
```

以上这三种规则结构，是今后我们要写的更复杂的选择组合的基础。更复杂的选择组合是啥意思？比如说，你想写一条 CSS 规则，希望给位于附注栏（aside 元素）中的一个段落添加样式，让这个段落看起来与正文（比如 article 元素）中的段落不一样。但我们到目前为止看到的规则，会影响整个文档中同种类型的所有元素。下面我们就来讲一讲怎么随心所欲地选择在标记中位于特殊区域的元素。

所有用于选择特定元素的选择符又分三种。

- **上下文选择符**。基于祖先或同胞元素选择一个元素。
- **ID 和类选择符**。基于 id 和 class 属性的值（你自己设定）选择元素。
- **属性选择符**。基于属性的有无和特征选择元素。

2.2　上下文选择符

下面，我们就来解决上一节提到的那个问题：你想分别给 article 和 aside 中的段落文本，分别设置不同的字号。像这种“基于位置”变换某个标签样式的问题，可以用上下文选择符来解决，本节我们就来讲一讲这种选择符。

上下文选择符的格式如下：

标签 1 标签 2 {声明}

其中，标签 2 就是我们想要选择目标，而且只有在标签 1 是其祖先元素（不一定是父元素）的情况下才会被选中。

上下文选择符，严格来讲（也就是 CSS 规范里），叫后代组合式选择符（descendant combinator selector），就是一组以空格分隔的标签名。用于选择作为指定祖先元素后代的标签。

```
article p {font-weight:bold;}
```

这个例子中的上下文选择符表明，只有是 article 后代的 p 元素才会应用后面的样式。换句话说，上面这条规则的目标是位于 article 上下文中的 p 元素。

可以通过下面这段代码来更深入地理解上下文选择符的含义。

```
<body>
    <article>
        <h1>Contextual selectors are <em>very</em> selective</h1>
        <p>This example shows how to target a <em>specific</em> tag.</p>
    </article>
    <aside>
        <p>Contextual selectors are <em>very</em> useful!</p>
    </aside>
</body>
```

以上标记在没有应用 CSS 的情况下，如图 2-3 所示。



图 2-3 浏览器默认样式下的标记

以上标记创建的 DOM（文档对象模型）结构如图 2-4 所示。

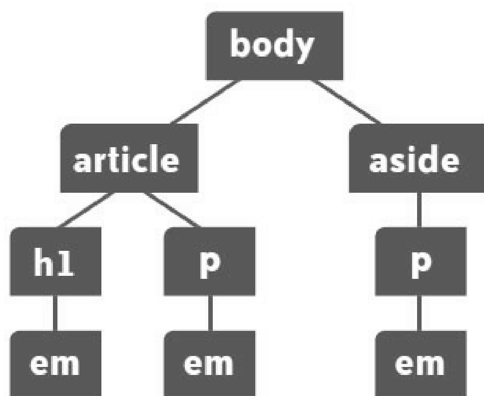


图 2-4　标记对应的 DOM 层次结构

图 2-5 展示了这个 DOM 结构对应的页面中盒子的嵌套关系。

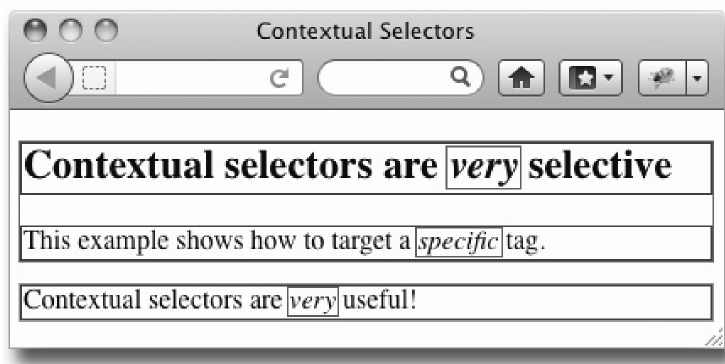


图 2-5　每一层结构都是一个包含子元素的盒子



图中盒子的轮廓是通过在 Firefox 的 Web Developer 工具条中选择 Outline Block Level Elements 和 Outline Custom Elements（可以在弹出的窗口中输入行内元素的名字）生成的。

在接下来我向大家展示那些瞄准标记中特定元素的 CSS 规则时，建议大家参照图 2-4 的 DOM 结构图。

我们就从最简单的一个标签的选择符开始。

```
em {color:green;}
```

这条规则选择页面中所有 em 元素，因此三个 em 元素中的文本都会变成绿色，如图 2-6 所示（图中为浅灰色）。

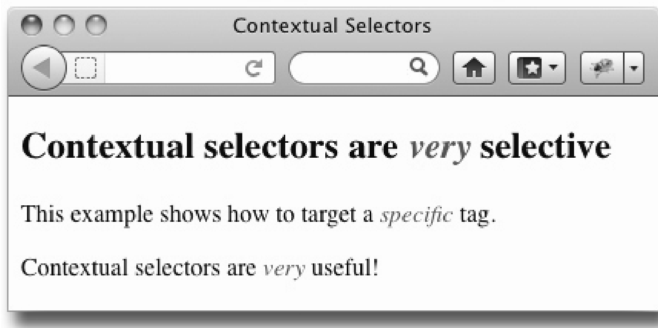


图 2-6　一个简单的单标签规则为所有 em 元素应用了绿色

接下来修改这个上下文选择符，让它选择页面中特定区域中的标签。

```
p em {color:green;}
```



万万牢记，上下文选择符以空格作为分隔符，而分组选择符则以逗号作为分隔符，不要弄混。

现在，前面的 p 元素是上下文，后面的 em 元素是要选择的目标。这里选择符的意思用白话说，就是：“选择以 p 元素为祖先的所有 em 元素。”好好，看看图 2-4 中的结构图，哪些标签会被这条规则选中？

没错，你说得对，会选中两个段落中的 em，而不会选中标题中的 em。因为标题中的 em 并没有作为祖先的段落，因此这条规则对它不适用（参见图 2-7）。



图 2-7　在以段落作为上下文的情况下，这条规则没有影响标题中的 em

图 2-8 进一步展示了单上下文选择符的可能性。下面的 CSS 规则

```
article em {color:green;}
```

会得到下面的结果



图 2–8　在以 article 作为上下文的情况下，aside 元素中的 em 没有受到影响

这次，我们使用了 article 作为上下文，因此该元素中标题及段落下的 em 都变成了绿色（图中是浅灰色）。而位于 aside 中的 em 未受影响，因为它没有 article 这么个祖先元素。注意，至于 article 与 em 之间还隔着一个 p 元素（或 h1 元素），都不影响结果。只要 em 在整个层次结构中有一个叫 article 的祖先元素就可以了。

要是我只想选择标题中的 em 呢？那就需要更加具体的上下文。

```
article h1 em {color:green;}
```

这里选择符的意思就是说："选中的 em 必须有一个祖先是 h1，后者进而还要有一个祖先是 article。"

如图 2-9 所示，为了选择某个非常具体的标签，有时候不得不串起一串上下文选择符。

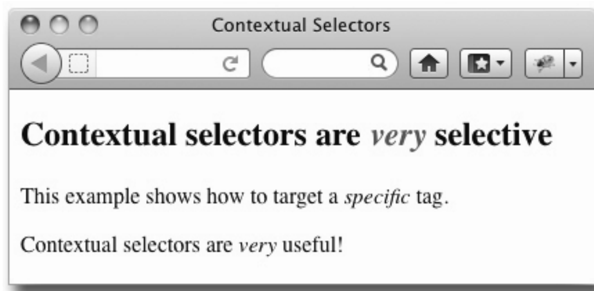


图 2–9　双上下文选择符可以选中更具体的标签

2.3 特殊的上下文选择符

刚才，我们学习的上下文选择符是以某个祖先标签作为上下文。只要有标签在它的层次结构“上游”存在这么一个祖先，那么就会选中该标签。无论从该标签到作为祖先的上下文之间隔着多少层次都没有关系。不过，有时候我们可能还会需要比“某些祖先”更加具体的上下文。比如说吧，要是你想根据父元素或者同胞元素的标签名来选择元素怎么办呢?

下面我们再用另一段标记来演示几种特殊的上下文选择符。

```
<section>
    <h2>An H2 Heading</h2>
    <p>This is paragraph 1</p>
    <p>Paragraph 2 has <a href="#">a link</a> in it.</p>
    <a href="#">Link</a>
</section>
```

2.3.1 子选择符>

标签 1 > 标签 2

标签 2 必须是标签 1 的子元素，或者反过来说，标签 1 必须是标签 2 的父元素。与常规的上下文选择符不同，这个选择符中的标签 1 不能是标签 2 的父元素之外的其他祖先元素。

```
section > h2 {font-style:italic;}
```

图 2-10 展示了以上规则的结果。

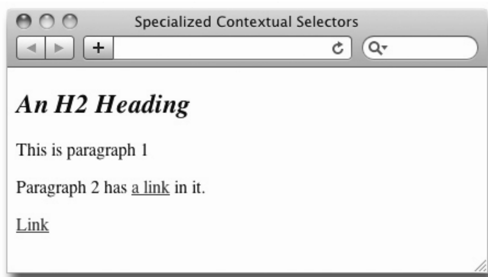


图 2-10 h2 是 section 的子元素，故而被选中

2.3.2　紧邻同胞选择符+

标签 1 + 标签 2

标签 2 必须紧跟在其同胞标签 1 的后面。

```
h2 + p {font-variant:small-caps;}
```

图 2-11 展示了这条规则的效果。

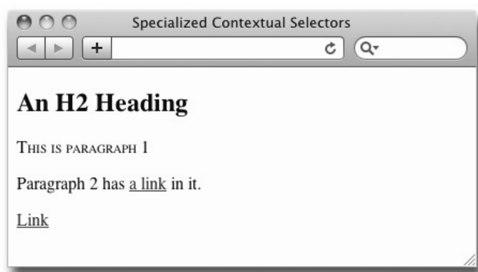


图 2–11　第一个 p 由于是 h2 的紧邻同胞而被选中

2.3.3　一般同胞选择符~

标签 1 ~ 标签 2

标签 2 必须跟（不一定紧跟）在其同胞标签 1 后面。



用 Shift+1 键左侧的键输入字符 ~（波浪号）。

```
h2 ~ a {color:red;}
```

图 2-12 展示了这条规则的效果。

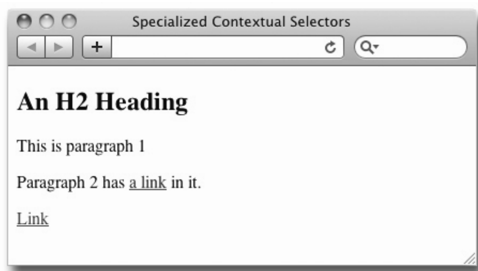


图 2–12　只选中了同胞元素

2.3.4　通用选择符*

*（按 Shift+8）

通用选择符*（常被称为星号选择符）是一个通配符，它匹配任何元素，因此下面这条规则

```
* {color:green;}
```

会导致所有元素（的文本和边框）都变成绿色。不过，一般在使用*选择符时，都会同时使用另一个选择符，比如：



color 属性设定的是前景色。前景色既影响文本也影响边框，但人们通常只用它设定文本颜色。

```
p * {color:red;}
```

这样只会把 p 包含的所有元素的文本变成红色。

这个选择符有一个非常有意思的用法，即用它构成非子选择符，比如：

```
section * a {font-size:1.3em;}
```

如图 2-13 所示，任何是 section 孙子元素，而非子元素的 a 标签都会被选中。至于 a 的父元素是什么，没有关系。

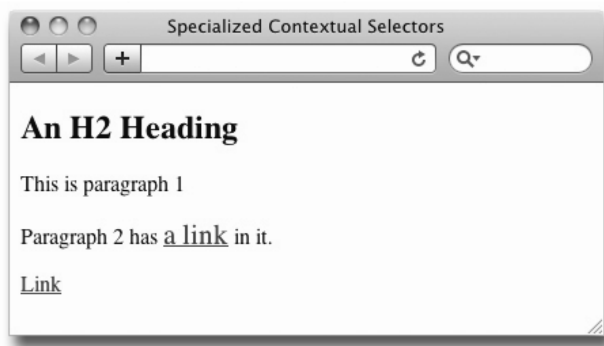


图 2-13　只选中了孙子元素，并未选中子元素

总之，只有一个标签名的选择符会选中页面中所有相同标签的实例。而通过上下文选择符，则可以指定标签必须具备相应的祖先或同胞。

2.4　ID 和类选择符

ID 和类为我们选择元素提供了另一套手段，利用它们可以不用考虑文档的层次结构。只要你在 HTML 标记中为元素添加了 id 和 class 属性，就可以在 CSS 选择符中使用 ID 和类名，直接选中文档中特定的区域。



可以给 id 和 class 属性设定任何值，但不能以数字或特殊符号开头。

2.4.1　类属性

类属性就是 HTML 元素的 class 属性，body 标签中包含的任何 HTML 元素都可以添加这个属性。下面这段代码展示了 HTML class 属性的用法。

```
<h1 class="specialtext">This is a heading with the <span>same class</span>
    as the second paragraph.</h1>
<p>This tag has no class.</p>
<p class="specialtext"> When a tag has a class attribute, you can target it
    <span>regardless</span> of its position in the hierarchy.</p>
```

图 2-14 展示了这段代码的显示结果。

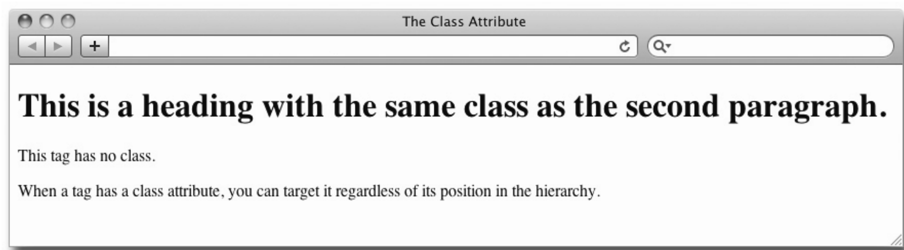


图 2-14　应用默认样式的效果

注意，我已经给其中两个标签添加了 specialtext 类。

1. 类选择符

.类名



注意，类选择符前面是点（.），紧跟着类名，两者之间没有空格。

类选择符就是在 HTML 类名前面加一个点（英文句号）。

好，我们来为标签应用如下 CSS 样式。

```
p {font-family:helvetica, sans-serif; font-size:1.2em;}
.specialtext {font-style:italic;}
```

图 2-15 展示了应用样式后的效果。

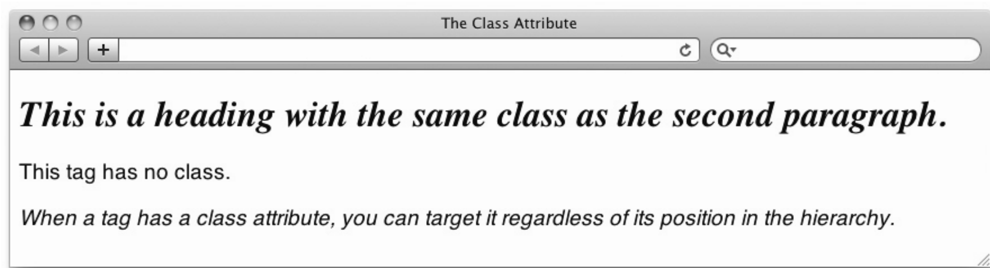


图 2-15 两个段落中文本的字体都变成了 Helvetica，而标题和第二个段落都有 specialtext 类，所以都变成了斜体

应用这两条规则的结果，就是两个段落都以 Helvetica 字体（如果本地没有该字体则使用浏览器通用的无衬线字体）显示，而带有 `specialtext` 类的段落同时也变成了斜体。`h1` 中的文本仍然使用浏览器默认字体（一般是 Times），因为 Helvetica 字体只应用给段落，不过由于它也有 `specialtext` 类，所以也应用了斜体。另外，没有默认样式的 `span`，由于我们没有明确为其添加样式，所以就继承了其父元素的样式。

2. 标签带类选择符

如果你只想瞄准带有这个类的段落，可以把标签名和类选择符写在一块，比如：

```
p {font-family:helvetica, sans-serif; font-size:1.2em;}
.specialtext {font-style:italic;}
p.specialtext {color:red;}
```

如图 2-16 所示，第三条 CSS 规则只选择带 `specialtext` 类的段落。像这样组合标签名和类选择符，可以让你更精确地选择特定的标签。

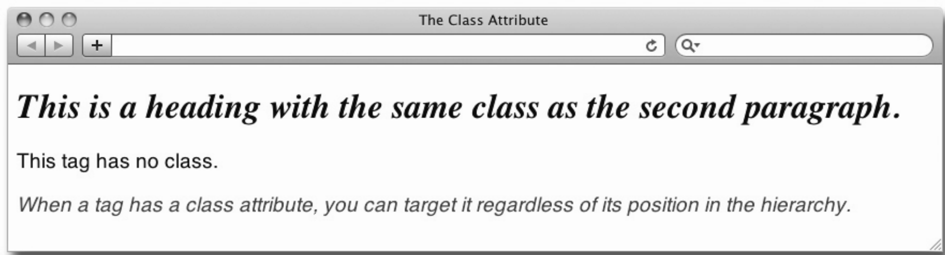


图 2-16　通过组合标签名和类选择符，可以让选择符更具体

下面我们就把这个技巧进一步发挥一下。

```
p {font-family:helvetica, sans-serif; font-size:1.2em;}
.specialtext {font-style:italic;}
p.specialtext {color:red;}
p.specialtext span {font-weight:bold;}
```

如图 2-17 所示，单词"regardless"变成了粗斜体，因为包含它的 span 位于一个带有 specialtext 类的段落中。这四条规则全都会对 span 产生影响，因为它从其父元素，那个带 specialtext 类的段落那里，继承了前三条规则的样式。我们已经两次提到了继承，关于继承我们放在本章后面再详细介绍。

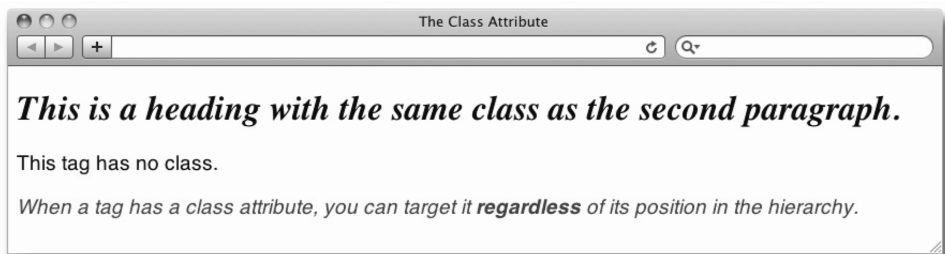


图 2-17　再添加一个选择符，可以把选择目标锁定为更特殊的元素

3. 多类选择符

可以给元素添加多个类，比如：

```
<p class="specialtext featured">Here the span tag <span>may or may not</span>
    be styled.</p>
```

多个类名，如这里的 specialtext 和 featured，放在同一对引号里，用空格分隔。实际上，更准确的说法，就应该是 HTML 的 class 属性可以有多个空格分隔的值。要

选择同时存在这两个类名的元素，可以这样写：

```
.specialtext.featured {font-size:120%;}
```

注意，CSS 选择符的两个类名之间没有空格，因为我们只想选择同时具有这两个类名的那个元素。如果你加了空格，那就变成了“祖先/后代”关系的上下文选择符了。

每个类名分别用一个 HTML `class` 属性的做法是常见的错误，正确的做法是像上面的代码那样，只用一个 `class` 属性，但给它设定多个值。本章后面还会讲到多类名的实际用法。

2.4.2 ID属性

ID 与类的写法相似，而且表示 ID 选择符的#（井号）的用法，也跟表示类选择符的.（句号）类似。

如果有一个段落像下面这样设定了 ID 属性

```
<p id="specialtext">This is the special text.</p>
```

那么，相应的 ID 选择符就是这样的：

```
#specialtext {CSS 样式声明}
```

或者这样的：

```
p#specialtext {CSS 样式声明}
```

除此之外，ID 与类的用法都一样，而且我们前面讨论的关于类选择符的（几乎）一切，都适应于 ID 选择符。那两者到底有什么区别呢？

用于页内导航的 ID

ID 也可以用在页内导航链接中。下面就是一个链接，其目标是同一页的另一个位置。

```
<a href="#bio">Biography</a>
```

看到 href 属性值开头的#了吗？它表示这个链接的目标在当前页面中，因而不会触发浏览器加载页面（如果没有#，浏览器就会尝试加载 bio 目录下的默认页面了）。

使用与 CSS 选择符里相同的 #ID 名语法，可以把链接导航到同一页面中的目标 ID。在这个页面的下方，应该有对应的目标元素。

```
<h3 id="bio">Biography</h3>
<p>I was born when I was very young…</p>
```

同样要注意，作为目标的 ID 值前面是没有#的，就是一个普通的 ID 值。

用户单击前面的链接时，页面会向下滚动到 ID 值为 bio 的 h3 元素的位置。如果链接的 href 属性里只有一个#，那么点击该链接会返回页面顶部。

```
<a href="#">Back to Top</a>
```

换句话说，要写一个“返回顶部”链接，根本不需要 ID 为#的目标元素。

另外，如果你暂时不知道某个 href 应该放什么 URL，也可以用#作为占位符，但不能把该属性留空。因为 href 属性值为空的链接的行为跟正常链接不一样。这样，团队中的其他人将来可以用中间层（比如 PHP）变量替换#，以便动态接收来自数据库的 URL。

2.4.3　什么时候用ID，什么时候用类

乍一看，类和 ID 都是用于在标记中标识特定标签的 HTML 属性，似乎完全是可以相互取代嘛。然而，它们的用途实际上大不相同。

1. 什么时候使用 ID

ID 的用途是在页面中唯一地标识一个元素。正因为如此，同一个页面中的每一个 ID 属性，都必须有独一无二的值（名字）。好吧，换一个角度讲，每个 ID 名在页面中都只能用一次。

```
<nav id="mainmenu">
```



也可以使用 ID 把 JavaScript 与某个标签关联起来（比如，当用户鼠标移动到一个链接上面时，运行激活动画的脚本）。ID 值的唯一性对 JavaScript 尤其重要，否则就会导致 JavaScript 行为异常。

在这里，页面中就不能再有其他元素使用 mainmenu 作为 ID 名了。为了标识页面的某一部分，比如主导航菜单，可以为 nav（navigation，导航）添加一个 ID 属性，并让

它包含菜单元素。

```
<nav id="mainmenu">
    <ul>
        <li><a href="#">Yin</a></li>
        <li><a href="#">Yang</a></li>
    </ul>
</nav>
```

有了用唯一 ID 标识的菜单之后，就可以使用上下文选择符来选择其中包含的各种类型的标签了。比如，可以将这个菜单中的链接设置为橙色，同时又不会影响页面中的其他链接：

```
#mainmenu a {color:orange;}
```

利用唯一 ID，可以在 CSS 中方便地定位到这个元素，以及它的子元素。到了后面读者会发现，我经常会给页面中每个顶级区域都添加一个 ID，从而得到非常明确的上下文，以便编写 CSS 时只选择嵌套在相应区域内的标签。

差不多了吧，你已经理解了 ID 表示的是页面中一个唯一的 HTML 元素，下面就来聊一聊什么时候使用类吧。

2. 什么时候使用类

类的目的是为了标识一组具有相同特征的元素，比如本章前面例子中的那个 specialtext 类。

在下面这个孩子名字的列表中，我想把男孩的名字变成蓝色，把女孩的名字变成粉红色。首先，我用类在标记中标识出了性别。

```
<nav>
    <ul>
        <li class="boy"><a href="#">Alan</a></li>
        <li class="boy"><a href="#">Andrew</a></li>
        <li class="girl"><a href="#">Angela</a></li>
        <li class="boy"><a href="#">Angus</a></li>
        <li class="girl"><a href="#">Anne</a></li>
        <li class="girl"><a href="#">Annette</a></li>
    </ul>
</nav>
```

然后，再用 CSS 为链接应用颜色：

```
.boy a {color:#6CF;}/*蓝色*/
.girl a {color:#F9C;}/*粉红色*/
```

第一条规则选择所有类名为 boy 的祖先元素包含的 a 元素，第二条规则选择所有类名为 girl 的祖先元素包含的 a 元素。这两种情况下的祖先元素，都是作为相应链接父元素的 li 元素。

不要乱用类

要避免 Web 开发专家 Jeffrey Zeldman 说的“类泛滥——标记中的麻疹”出现。什么意思呢？就是说你不要像使用 ID 一样，每个类都指定一个不同的类名，然后再为每个类编写规则。如果你确实有这种随意使用类的习惯，那说明你可能像大多数对 CSS 充满激情的初学者一样，还不了解继承和上下文选择符的作用。于是，你可能会给每个标签都重复写同样的样式（比如为页面中很多标签分别指定相同的字体）。实际上，继承和上下文选择符能让不同的标签共享样式，从而降低你需要编写和维护的 CSS 量。

2.4.4　ID和类的小结

ID 的用途是在页面标记中唯一地标识一个特定的元素。它能够为我们编写 CSS 规则提供必要的上下文，排除无关的标记，而只选择该上下文中的标签。

相对来说，类是可以应用给任意多个页面中的任意多个 HTML 元素的公共标识符，以便我们为这些元素应用相同的 CSS 样式。而且，使用类也让为不同标签名的元素应用相同的样式成为可能。

2.5　属性选择符

到目前为止，我们已经学习了使用上下文选择符、ID 和类选择 HTML 元素。再有一种选择元素的方法是属性选择符，它基于 HTML 标签的属性选择元素。以下是两个常见的例子。

2.5.1 属性名选择符

标签名[属性名]

选择任何带有属性名的标签名。

比如，下面的 CSS

```
img[title] {border:2px solid blue;}
```

会导致像下面这个带有 `title` 属性的 HTML `img` 元素显示 2 像素宽的蓝色边框，至于 `title` 属性有什么值，无关紧要，只要有这个属性在就行啦。

```
<img src="images/yellow_flower.jpg" title="yellow flower" alt="yellow flower" />
```

什么情况下会用到这个属性选择符呢？比如，可以在用户鼠标移动到这些图片上时，此时浏览器会显示一个（利用 `title` 属性中的文本生成的）提示条。一般来说，人们经常给 `alt` 和 `title` 属性设定相同的值。`alt` 属性中的文本会在图片因故未能加载时显示，或者由屏幕阅读器朗读出来。而 `title` 属性会在用户鼠标移动到图片上时，显示一个包含相应文本的提示。

2.5.2 属性值选择符

标签名[属性名="属性值"]



在 HTML5 中，**属性值**的引号可加可不加，在此为了清楚起见，我们加了。

选择任何带有值为属性值的属性名的标签名。

这个选择符可以让你控制到属性的值是什么。例如，这条规则

```
img[title="red flower"] {border:4px solid green;}
```

在图片的 `title` 属性值为 `red flower` 的情况下，才会为图片添加边框。换句话说，下面这个 `img` 元素就会被加上边框。

```
<img src="images/red_flower.jpg" title="red flower" alt="red flower" />
```

当然，属性选择符还有其他形式，感兴趣的话，就请访问 http://www.stylinwith-css.com 吧。

2.5.3 属性选择符的小结

基于属性名和属性的其他特征选择元素，为我们提供了另一种区别对待相同标签的机会。只要事先规划好，就可以编写出适合属性选择符的标记来。

到现在为止，我们介绍的选择符都有一个共同点，即它们针对的都是标记中的某个部分，比如标签名、类名、ID、属性或属性值。然而，使用 CSS 还可以在某些事件发生时，改变某些元素的样式，比如用户鼠标悬停在一个链接上。而这就要靠伪类来实现了。

2.6 伪类

伪类这个叫法源自它们与类相似，但实际上并没有类会附加到标记中的标签上。伪类分两种。

- UI（User Interface，用户界面）伪类会在 HTML 元素处于某个状态时（比如鼠标指针位于链接上），为该元素应用 CSS 样式。
- 结构化伪类会在标记中存在某种结构上的关系时（如某个元素是一组元素中的第一个或最后一个），为相应元素应用 CSS 样式。

2.6.1 UI伪类

UI 伪类会基于特定 HTML 元素的状态应用样式。最常使用 UI 伪类的元素是链接（a 元素），利用 UI 伪类，链接可以在用户鼠标悬停时改变文本颜色，或者去掉文本的下划线。此外，还可以有其他响应方式，比如悬停时显示一个信息面板，相关内容在我们讨论交互组件的时候再聊。

1. 链接伪类

针对链接的伪类一共有 4 个，因为链接始终会处于如下 4 种状态之一。

- Link。此时，链接就在那儿等着用户点击。
- Visited。用户此前点击过这个链接。
- Hover。鼠标指针正悬停在链接上。

- Active。链接正在被点击（鼠标在元素上按下，还没有释放）。

以下就是这些状态对应的 4 个伪类选择符（使用了 a 选择符和一些示例声明）：

```
a:link {color:black;}
a:visited {color:gray;}
a:hover {text-decoration:none;}
a:active {color:red;}
```



由于这 4 个伪类的特指度（本章后面再讨论特指度）相同，如果不按照这里列出的顺序使用它们，浏览器可能不会显示预期结果。为了好记，我建议大家可以这么想："LoVe? HA!"大写字母就是每个伪类的头一个字母。

选择符中与众不同的:（冒号）好像在向我们宣示："我是一个伪类！"



一个冒号（:）表示伪类，两个冒号（::）表示 CSS3 新增的伪元素。尽管浏览器目前都支持对 CSS 1 和 CSS 2 的伪元素使用一个冒号，但希望你能习惯于用双冒号代替单冒号，因为这些单冒号的伪元素最终可能都会被淘汰掉。更多相关信息，可以参见这里：http://www.w3.org/TR/2005/WD-css3- selectors-20051215/#pseudo-elements。

根据前面的声明，链接在初始状态时是黑色（默认带下划线）。当鼠标移到上面时（悬停状态），链接的下划线消失，颜色仍然是黑色。当用户在链接上按下鼠标时（活动状态），链接变成红色。而在链接被点击后，也就是鼠标在链接上按下，又在链接上释放后，会触发浏览器打开 URL，此后（或者更准确地说，到浏览器访问历史中的这个 URL 过期或被用户删除之前），链接会一直显示为灰色。

不一定非得把这 4 个状态都写出来。如果你只想定义`:link`和`:hover`状态，没问题，大多数情况下这也足够了。但如果你有一个长长的目录链接，那么用稍浅一些的颜色显示出那些已经访问过（即点击过）的链接，对用户会很有帮助。然而，修改导航条`visited`状态的颜色就没有什么意义了。

我一般就定义一个 a 状态和一个`:hover`状态，后者就是为了让用户（在鼠标悬停时）知道相应的元素是不是可以点。

以这种方式为链接的不同的状态添加样式是不错，但要发挥这些链接伪类的真正威力，还是得把它们用到上下文选择符中。这样，就可以让不同区块内的链接具有不同的外观和行为了。比如，稍后我们会看到，通过在上下文选择符中使用链

接伪类，可以轻易地为 nav、footer、aside 和 article 元素中的链接应用不同的外观和行为。

注意，有些伪类可以用于任何元素，而不仅仅是 a 元素。比如，下面这条规则能让段落背景在鼠标悬停时变成灰色：

```
p:hover {background-color:gray;}
```

2. :focus 伪类

e:focus



在这个以及后续的例子中，e 表示任何元素，如 p、h1、section，等等。

表单中的文本字段在用户单击它时会获得焦点，然后用户才能在其中输入字符。下面的规则

```
input:focus {border:1px solid blue;}
```

会在光标位于 input 字段中时，为该字段添加一个蓝色边框。这样可以让用户明确地知道输入的字符会出现在哪里。

3. :target 伪类

e:target

如果用户点击一个指向页面中其他元素的链接，则那个元素就是目标（target），可以用:target 伪类选中它。

对于下面这个链接

```
<a href="#more_info">More Information</a>
```

位于页面其他地方、ID 为 more_info 的那个元素就是目标。该元素可能是这样的：

```
<h2 id="more_info">This is the information you are looking for.</h2>
```

那么，如下 CSS 规则

```
#more_info:target {background:#eee;}
```

会在用户单击链接转向 ID 为 more_info 的元素时，为该元素添加浅灰色背景。

维基百科在其引证中大量使用了:target 伪类。维基百科的引证链接就是正文里那些不起眼的数字链接。引证本身则位于长长的页面的最下方。如果没有:target 应用的突出显示，很难知道你点击的链接对应着一大堆引证中的哪一个。

2.6.2 结构化伪类

结构化伪类可以根据标记的结构应用样式，比如根据某元素的父元素或前面的同胞元素是什么。

1.:first-child 和:last-child

e:first-child
e:last-child

:first-child 代表一组同胞元素中的第一个元素，而:last-child 则代表最后一个。比如，把下面的规则

```
ol.results li:first-child {color:blue;}
```

应用给以下标记：

```
<ol class="results">
    <li>My Fast Pony</li>
    <li>Steady Trotter</li>
    <li>Slow Ol' Nag</li>
</ol>
```

文本“My Fast Pony”就会变成蓝色。如果选择符改成这样：

```
ol.results li:last-child {color:red;}
```

那变成红色的文本就是“Slow Ol’ Nag”了。

2.:nth-child

e:nth-child(n)



e 表示元素名，n 表示一个数值（也可以使用 odd 或 even）。

例如，

```
li:nth-child(3)
```

会选择一组列表项中的每个第三项。

`:nth-child` 伪类最常用于提高表格的可读性，比如像第 6 章中那样，对表格的所有行交替应用不同颜色。

还有其他一些结构化伪类，完整的信息请参考这里：http://www.stylinwithcss.com。

2.7　伪元素

顾名思义，伪元素就是你的文档中若有实无的元素。以下我们介绍几个最有用的伪元素，其他伪元素请参考 http://www.stylinwithcss.com。

1. `::first-letter` 伪元素

e::first-letter

比如，以下 CSS 规则：

```
p::first-letter {font-size:300%;}
```

可以得到如图 2-18 所示的段落首字符放大的效果。



如果不用伪元素创建这个首字符放大效果，必须手工给该字母加上<span>标签，然后再为该标签应用样式。而伪元素实际上是替我们添加了无形的标签。

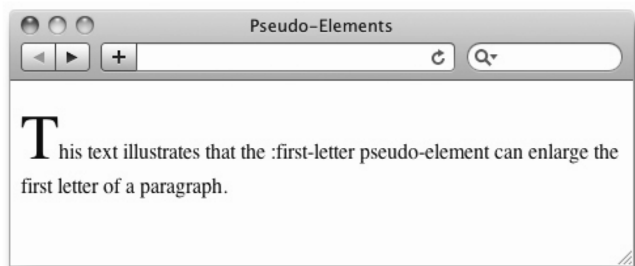


图 2-18　`::first-letter` 伪元素可以用来创建首字符放大效果

2. `::first-line` 伪元素

e::first-line

可以选中文本段落（一般情况下是段落）的第一行。例如

```
p::first-line {font-variant:small-caps;}
```

可以把第一行以小型大写字母显示，见图 2-19。

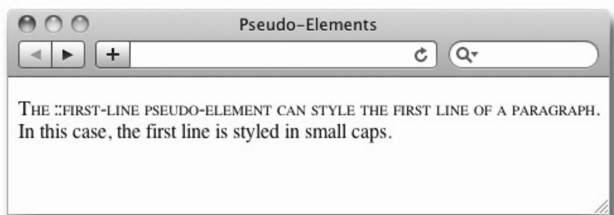


图 2-19 用::first-line 伪元素把第一行变成了小型大写字母。注意，::first-line 伪元素的长度会随浏览器窗口大小的变化而改变。

3.::before 和::after 伪元素

以下两个伪元素

```
e::before
e::after
```

可用于在特定元素前面或后面添加特殊内容。

以下标记

```
<p class="age">25</p>
```

和如下样式

```
p.age::before {content:"Age: ";}
p.age::after {content:" years.";}
```



注意，每个 content 属性值中都包含了空格，以便输出结果中有适当的空距。

能得到以下结果：

```
Age: 25 years.
```

如果标签中的内容是通过数据库查询生成的结果，那么用这种技巧再合适不过了。因为所有结果都是数字，使用这两个伪元素可以在把数字呈现给用户时，加上说明性文字。

这个例子展示了对::before 和::after 伪元素很基本又很实用的应用。后面我还会给

大家展示这两个伪元素的其他用法，比如在应用它们的元素外面附着一个动态的新元素，从而得到一种有趣的布局效果。



搜索引擎不会取得伪元素的信息（因为它在标记中并不存在）。因此，不要通过伪元素添加你想让搜索引擎索引的重要内容。

好啦，到现在为止，我们已经把各种 CSS 选择符都介绍完了。接下来，需要更进一步，探讨一下 CSS 的工作原理。

在一个较大的样式表中，可能会有很多条规则都选择同一个元素的同一个属性。比如，一个带有类属性的段落，可能会被一条以标签名作选择符的规则选中并指定一种字体，而另一条以该段落的类名作选择符的规则却会给它指定另一种字体。我们知道，字体属性在任意时刻都只能应用一种设定，那此时该应用哪种字体呢？为解决类似的冲突，确定哪条规则"胜出"并最终被应用，CSS 提供了三种机制：继承、层叠和特指。接下来的三节，就分别讨论这三种机制。

2.8　继承

就像你总是指望着腰缠万贯的老爸有一天会留给你一大笔遗产一样，CSS 中的祖先元素也会向后代传递一样东西：CSS 属性的值。还记得吗，我们在第 1 章讲文档层次结构时提到过，body 是所有元素的老祖宗，所有标签都是它的后代。那么由于 CSS 继承的约定，如果我们为 body 像下面一样写一条规则

```
body {font-family:helvetica, arial, sans-serif;}
```

那么，文档中的所有元素，无论它在层次结构中多么靠下，都将继承这些样式，以 Helvetica 字体（或者在 Helvetica 字体无效时以其他字体代替）显示各自包含的文本。继承给我们带来的效率是显而易见的，全站的主字体只要在某个上层元素上指定即可，无须在每一个标签上分别指定。而对于个别想使用不同字体的元素，只要个别设定 font-family 属性就好了。

CSS 中有很多属性是可以继承的，其中相当一部分都跟文本有关，比如颜色、字体、字号。然而，也有很多 CSS 属性不能继承，因为继承这些属性没有意义。这些不能继承的属性主要涉及元素盒子的定位和显示方式，比如边框、外边距、内边距，这些下一章我们才会讲到。

举个例子吧，假设我们想创建一个边栏，在其中放一组链接。为此，我们用 `nav` 元素嵌套该组链接，并给 `nav` 应用了一种字号和一个边框效果，比如 2 像素宽的红色边框。不难想象，`nav` 中的所有链接都继承它的字号很正常，可要是也继承它的边框就不合适了。当然，这些链接不会继承边框效果，因为 `border` 属性不能继承。

由于字体和文本样式是可以继承的，所以在使用相对字体单位（如百分比和 `em`）时要格外小心。如果某个标签的字体大小被设置为 80%，而它的一个后代的字体大小也被设置为 80%，那么该后代中文本最终的字体大小将是 64%（80%的 80%）。这有时候可能并不是你想要的结果。第 4 章我们会详细介绍绝对和相对大小的利弊。

本书后面的例子还会继续向大家示范继承的作用，展示如何利用继承以最少的 CSS 代码达成预期效果。

2.9　层叠

好啦，大家现在的背景知识已经足够理解层叠了。层叠，就是层叠样式表中的层叠，是一种样式在文档层次中逐层叠加的过程，目的是让浏览器面对某个标签特定属性值的多个来源，确定最终使用哪个值。

层叠是 CSS 的核心机制，理解了它才能以最经济的方式写出最容易改动的 CSS，让文档外观在达到设计要求的同时，也给用户留下一些空间，让他们能根据需要更改文档的显示效果（比如整体调整字号）。

2.9.1　样式来源

样式有多处来源。首先，如果告诉你浏览器有一个默认的样式表，你应该不会感到奇怪，因为你还没有写一行 CSS 呢，每个标签已经带了一定的样式。`h1` 是不是粗体，字号还挺大？`em` 是不是斜体？列表呢，是不是缩进而且还带项目符号或编号？

然后，有一个用户样式表。没错，用户也可以提供样式表，尽管这样的用户不多见。这个选择对于视障用户很有用，他们可以通过用户样式表，强制浏览器加载的所有网站都以更大的字号，更容易分辨的颜色显示内容。比如，某个视障用户可以增加如下样式：

```
body {font-size:200%;}
```

这样就把字号调大了一倍，继承同样也会起作用。

再有，就是作者样式表，也就是网页设计师（你）写的样式表。前面我们已经讲了作者给网页添加样式的三种方法：链接样式、嵌入样式和行内样式。



参见 2.1 节中的“为文档添加样式的三种方法”。

以下就是浏览器层叠各个来源样式的顺序：

- 浏览器默认样式表
- 用户样式表
- 作者链接样式表（按照它们链接到页面的先后顺序）
- 作者嵌入样式
- 作者行内样式

浏览器会按照上述顺序依次检查每个来源的样式，并在有定义的情况下，更新对每个标签属性值的设定。整个检查更新过程结束后，再将每个标签以最终设定的样式显示出来。

举例来说，如果作者的链接样式表将 p 的字体设定为 Helvetica，而页面中有一条嵌入规则以相同的选择符把字体设定为 Verdana，那么段落文本最终会以 Verdana 字体显示。因为浏览器是在读取链接样式表之后读取嵌入样式。但要是用户和作者样式表都没有为段落指定字体，就会使用浏览器默认样式表中指定的 Times。

在接下来了解了层叠规则，知道了具体如何确定给某个页面元素应用何种样式之后，你对层叠的理解就会更加透彻。

2.9.2　层叠规则

以下是层叠机制的相关规则。



要了解有关层叠的更多信息，请参考这个链接：http://www.w3.org/TR/CSS2/cascade.html。

层叠规则一：找到应用给每个元素和属性的所有声明。浏览器在加载每个页面时，都会据此查到每一条 CSS 规则，标识出所有受到影响的 HTML 元素。

层叠规则二：按照顺序和权重排序。浏览器依次检查 5 个来源，并设定匹配的属性。如果匹配的属性在下一个来源也有定义，则更新该属性的值，如此循环，直到检查完页面中所有标签受影响属性的全部 5 个来源为止。最终某个属性被设定成什么值，就用什么值来显示。

声明也可以有权重。可以像下面这样为单独的声明增加权重：

```
p {color:green !important; font-size:12pt;}
```

空格!important 分号（;）用于加重声明的权重。

这条规则加重了将文本设置为绿色的权重。于是，就算层叠的下一来源给段落设定了其他颜色，最终的颜色值仍然还是绿色。说到底，就是一种特权，相当于你下了命令：就应用这个样式啦，其他来源一概不用考虑。不过，在使用这个特权之前，一定要知道你的这个!important 声明，很可能会让用户的个人设定不起作用，而用户的设定对他可能还非常重要。每次当你有使用这个特权的冲动时，最好先静下心来仔细分析一下自己的 CSS，多数情况下都应该可以想出一种更好的替代方案。就我个人而言，我是基本上不用!important 声明的。

层叠规则三：按特指度排序。除了有点拗口之外，特指度（specificity）其实表示一条规则有多明确。如果没有特指度的考量，那为了让恰当的样式起作用，恐怕我们就免不了要频繁变换样式表中规则的顺序了。

我们知道，如果某个样式表中包含如下规则：

```
p {font-size:12px;}
p.largetext {font-size:16px;}
```

那么下面的段落

```
<p class="largetext">A bit of text</p>
```

将显示 16 像素高的文本，因为第二条规则的选择符既包含标签名，也包含类名，所以意义更明确（特指度更高），结果第二条规则会覆盖第一条规则中的同名属性。这个例子似乎太明显了，如果还是对同一个段落，有如下样式呢？

```
p {font-size:12px;}
.largetext {font-size:16px;}
```

答案是尽管两条规则都匹配同一个标签，但使用类选择符的规则胜出，文本还是 16 像素高。为什么呀？因为类名选择符比普通的标签选择符具有更高的特指度。一条规则的特指度，由它的选择符中包含多少个标签、类名和 ID 决定。

2.9.3　计算特指度

下面我们具体讲一讲怎么计算选择符的特指度。首先，有一个简单的记分规则，即对每个选择符都要按下面的“ICE”公式计算三个值：

I - C - E



ICE 并非真正的三位数，只不过大多情况下把结果看成一个三位数没有问题，三位数最大的胜出。但是，千万得知道 0-1-12 与 0-2-0 相比，仍然是 0-2-0 的特指度更高。

三个字母间的短横线是分隔符，并非减号。针对这个公式的计分办法如下：

1. 选择符中有一个 ID，就在 I 的位置上加 1；
2. 选择符中有一个类，就在 C 的位置上加 1；
3. 选择符中有一个元素（标签）名，就在 E 的位置上加 1；
4. 得到一个三位数。

好了，下面通过几个例子来理解特指度。

```
P                               0-0-1 特指度=1
p.largetext                     0-1-1 特指度=11
p#largetext                     1-0-1 特指度=101
body p#largetext                1-0-2 特指度=102
body p#largetext ul.mylist      1-1-3 特指度=113
body p#largetext ul.mylist li   1-1-4 特指度=114
```

在此，每个选择符都比前一个选择符的特指度更高。

层叠规则四：顺序决定权重。如果两条规则都影响某元素的同一个属性，而且它们的特指度也相同，则位置最靠下（或后声明）的规则胜出。①

① 建议大家参考本书一位读者“流年”的文章“CSS 优先级特性”：http://liunian.info/css-specificity.html。
——译者注

嗯，亲爱的读者，层叠的概念确实不太好理解。特别是，如果你的 CSS 经验还不够多，那理解起来就更不容易了。不过，我总结了一个简化版的层叠规则（见下面“查理版简单层叠要点”），不仅适用于任何情况，而且也更容易记住。

查理版简单层叠要点

在这个查理版里，只要记住三条规则就够了。这三条规则适合所有情况。

规则一：包含 ID 的选择符胜过包含类的选择符，包含类的选择符胜过包含标签名的选择符。

规则二：如果几个不同来源都为同一个标签的同一个属性定义了样式，行内样式胜过嵌入样式，嵌入样式胜过链接样式。在链接的样式表中，具有相同特指度的样式，后声明的胜过先声明的。

规则一胜过规则二。换句话说，如果选择符更明确（特指度更高），无论它在哪里，都会胜出。

规则三：设定的样式胜过继承的样式，此时不用考虑特指度（即显式设定优先）。下面简单解释一下规则三。比如下面的标记

```
<div id="cascade_demo">
  <p id="inheritance_fact">Inheritance is <em>weak</em> in the Cascade</p>
</div>
```

和下面的规则

```
div#cascade_demo p#inheritance_fact {color:blue;}
2 - 0 - 2  （高特指度）
```

会导致单词“weak”变成蓝色，因为它从父元素 p 那里继承了这个颜色值。

但是，只要我们再给 em 添加一条规则

```
em {color:red;}
0 - 0 - 1  （低特指度）
```

em 就会变成红色。因为，虽然它的特指度低（0–0–1），但 em 继承的颜色值，会被为它明确（显式）指定的颜色值覆盖，就算（隐式）遗传该颜色值的规则的特指度高（2–0–2）也没有用。

2.10　规则声明

前面，我们主要介绍了怎么使用 CSS 规则的选择符选择标签，但对 CSS 规则的另一部分——声明，还一点都没有讨论过呢。为了介绍选择符，前面那些例子中曾使用过很多种声明，但始终都没有机会解释它们。好了，现在是时候了，本节专门讨论规则声明。

本章开头的图 2-2 展示了 CSS 规则的构成。我们也知道一个声明包含两部分：属性和值。属性指出要影响元素的哪个方面（颜色、高度，等等），而值表示把属性设定为什么（绿色、12px，等等）。

每个元素都有很多属性，可以通过 CSS 来设定。但每个元素具体有哪些属性，也会因元素而异。比如，可以给文本元素设定 `font-size` 属性，但图片则没有这个属性。后续几章会陆续讲到各种 HTML 元素的属性，但 CSS 属性的值则只有少数几种，所以我们现在就可以学习。

CSS 属性值主要分以下三类。

文本值。例如，`font-weight:bold` 声明中的 `bold` 就一个文本值。文本值也叫做关键字。

数字值。数字值后面都有一个单位，例如英寸或点。在声明 `font-size:12px` 中，`12` 是数字值，而 `px` 是单位（像素）。如果数字值为 `0`，那么就不用带单位了。

颜色值。颜色值可以用几种不同的格式来写，包括 RGB（Red, Green, Blue，红绿蓝）、HSL（Hue, Saturation, Luminance，色相，饱和度，亮度）和十六进制值（例如 `color:#336699`）。

下面我们依次介绍一下这三种值。

2.10.1　文本值

所有 CSS 属性都有文本值。例如，`visibility` 属性有 `visible` 和 `hidden` 值，`border-style` 属性有 `solid`、`dashed` 以及 `inset` 值。

除了上面两个属性外，还有很多属性可以使用文本值，但那些值都是特定于属性的。

所以，我会在后面几章中用到相应的属性时，再跟大家介绍它们的文本值。与文本值不同，数字值和颜色值的书写方式是有限的。

2.10.2 数字值

数字值用于描述元素的各种长度（在 CSS 里，“长度”的含义比较广，还包括高度、宽度、粗细，等等）。数字值主要分两类：绝对值和相对值。

表 2-1 列出了绝对值。绝对值描述的是一个真实的长度（比如，6 英寸），而相对值则是相对于其他基准的描述（比如“是某某的两倍长”）。

表2-1 绝对值及示例

绝 对 值	单位缩写	示 例*
英寸	in	`height:6in`
厘米	cm	`height:40cm`
毫米	mm	`height:500mm`
点	pt	`height:60pt`
皮卡	pc	`height:90pc`
像素	px	`height:72px`

* 示例值并不是相等的长度。

对于绝对单位，本书的例子中只使用像素，我在工作也是一样。但打印样式表是个例外，因为打印纸是以英寸为单位度量的。以相同的单位设计打印布局是最合适的。

虽然绝对单位的含义不需要什么解释，但表 2-2 中列出的相对单位则需要多花点笔墨才能讲清楚。

表2-2 相对值及示例

相 对 值	单位缩写	示 例*
Em	em	`height:1.2em`
Ex	ex	`height:6ex`
百分比	%	`height:120%`

* 示例值并不是相等的长度。

em 和 ex 都是字体大小的单位，但在 CSS 中，它们作为长度单位适用于任何元素。先说说 em，它表示一种字体中字母 M 的宽度，因此它的具体大小取决于你使用的字体。而 ex 呢，等于给定字体中字母 x 的高度（小写字母 x 代表一种字体的字母中间

部分的高度，不包括字母上、下突出的部分——如 d 和 p 上下都出头儿）。

百分比非常适合设定被包含元素的宽度，此时的百分比就是相对于宽度而言的。第 5 章会讲到，把 HTML 结构元素的宽度设定为 body 宽度的百分比，是“流式”设计的关键所在。这种布局设计可以随着用户调整浏览器窗口大小而成比例地伸缩。

2.10.3　颜色值

指定颜色值可以有几种方式。这些方式可以在同一个样式表中混合使用。

1. 颜色名（如 red）

就像前面讨论选择符的例子一样，设定颜色属性时可以直接使用颜色名，或者用官方术语就是颜色关键字。

W3C 定义了 16 个颜色关键字：aqua（浅绿色）、black（黑色）、blue（蓝色）、fuchsia（紫红色）、gray（灰色）、green（绿色）、lime（黄绿色）、 maroon（褐红色）、navy（深蓝色）、olive（茶青色 ）、purple（紫色）、red（红色）、silver（银色）、teal（青色）、white（白色）和 yellow（黄色）。要了解这些颜色名及其对应的 RGB 颜色值，请参考：http://www.w3.org/TR/css3-color/#html4。

大多数现代浏览器支持更多种颜色名（即 140 X11 颜色名），但如果真要使用颜色名，最好只使用前面那 16 种。要了解这 114 种颜色名及其对应的 RGB 颜色值，请参考：http://en.wikipedia.org/wiki/X11_color_names。

一般来说，颜色关键字最常用于指定白色和黑色。对于其他颜色真的很较真儿的场合，还得使用以下几种格式的颜色值。

2. 十六进制颜色（#RRGGBB 或#RGB）

如果你了解 C++、PHP 或 JavaScript，那么对用十六进制值表示颜色应该会比较熟悉。十六进制颜色的值的格式如下：

```
#rrggbb
```

例如橙色是：

```
#ff8800
```

千万别忘了值开头的#（井号)!

这个 6 位数的前两位定义红色，中间两位定义绿色，后两位定义蓝色。计算机使用二进制计数，而不是我们常用的十进制，因此才有了十六进制（16 是 2 的 4 次幂)。十六进制以 16 为基数，使用数字 0-9 和字母 a-f，共 16 个值。其中，a-f 代表 10 到 15。由于每种颜色用两位十六进制值表示，因此该颜色就有 256（16 × 16）种可能的值，结果就是 16 777 216（256 × 256 × 256）种组合，也就是可以表示那么多种颜色。

纯红色是#ff0000，纯绿色是#00ff00，而纯蓝色是#0000ff。



大多数十六进制颜色值不仔细分析可不容易猜，比如#7ca9be 是深蓝绿色，我怎么知道的？首先我们来看每一对 rgb 值中的第一个值，也就是 7、a、b。蓝色和绿色值相差无几，而红色值也没有那么深。有了这些信息，就可以大致猜出这个颜色了，对，是蓝绿色。

另外，如果三对值中的每一对是两个相同的数字，也可以使用简写形式：

```
#rgb
```



如今已经废弃的 216 种 Web 安全色，完全是由这种两两相同的颜色值构成的。

对于上面的例子来说，可以分别将它们写成#f00、#0f0 和#00f。再比如，#ff3322（深红色）可以简写成#f32。在设定阴影的时候，这种简写方式可以省不少劲儿。比如，#000 是纯黑，#444 是 75%灰色，#888 是 50%灰色，#bbb 是 25%灰色，而#fff 则是纯白色。

3. RGB 颜色值（R, G , B）

每种颜色都可以用一个 0 到 255（包含）之间的值指定。格式如下：

```
rgb(r, g, b)
```

比如，rgb(0,255,0)表示纯绿色。

没错，与十六进制 RGB 值一样，只不过使用的是十进制的数值。因为每种颜色都有 256 种可能，所以它能表示的颜色数量与十六进制格式的一样。不同的是，我们对十进制数更熟悉，毕竟是童子功嘛。而十六进制是计算机课上才学的（你在学校里学过计算机，对吧？）。

4. RGB 百分比值（R%, G%, B%）

这是用百分比来表示每种颜色值的一种方法，格式如下：

```
r%, g%, b%
```

可以接受的值是 0%到 100%。虽然以这种方法只能表示区区一百万（100 × 100 × 100）种颜色，但对我们绝大数人来说，也已经足够了。同样，使用百分比表示的颜色值，比使用十六进制更容易猜到你想要的颜色。

举个例子，100%, 0%, 0%是纯红色，0%, 100%, 0%是纯绿色，而 46%, 76%, 80%接近前面用十六进制值作例子分析的深蓝绿色。

5. HSL (色相, 饱和度%, 亮度%)

HSL 格式如下：

```
HSL(0,0%,0%)
```

HSL 比我们见过的各种 RGB 方式更直观，因为使用它更容易写出和看懂颜色。

HSL 颜色中的第一个值表示色相，也就是一个实际的颜色，比如红色和绿色。所有颜色围绕色相环（也叫色轮）一周，而色相值以圆周上的度数表示。

如图 2-20 所示，HSL 中的色相值以色轮上的度数表示（请参见本书封皮的前勒口的彩图）。

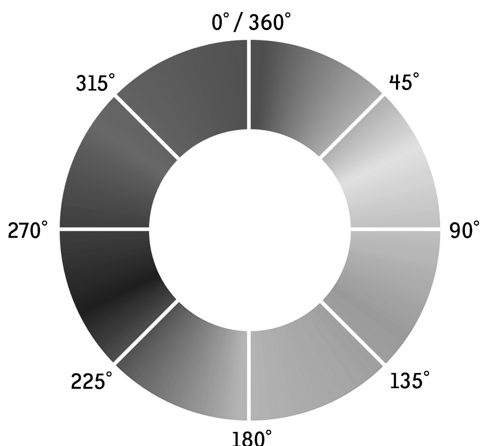


图 2-20　HSL 颜色模型中的色相值以圆周上的度数表示

红色是 0 或 360，青色是与之相对的 180。以下是彩虹七色在色轮中大致的色相值。

- 红：0
- 橙：35
- 黄：60
- 绿：125
- 蓝：230
- 靛：280
- 紫：305

至于饱和度和亮度，相对就容易理解多了。饱和度设定有多少颜色，灰色的饱和度低，而强烈的色彩饱和度高。亮度设定颜色的明暗，0%就是黑色，100%就是白色，而中间的值是实际能看到的色相。

如果把上面七彩虹的色相值都记住，或者就放在手边，那么你会发现想写出什么颜色都不在话下。RGB 和十六进制颜色值，都要求你事先在大脑里先混合颜色，而 HSL 则只有一个表示颜色的值。从把饱和度和亮度都设定为 50%，就可以轻松调制出你想要的任何颜色来。

6. Alpha 通道

请注意，RGB 和 HSL 都支持 Alpha 通道，用于设置颜色的不透明度（换句话说，就是能够透过多少背景）。相应的格式分别叫 RGBA 和 HSLA。其中，两种格式中的 A（alpha）值可以是 1（完全不透明）也可以是 0（完全透明），或者介于 1 和 0 之间的小数值。关于 Alpha 通道的详细内容，我们会在下一章再讲。

关于颜色的一些资料

- http://colrd.com：Colrd 是一个 Pinterest 风格的站点，其中有很多能启发人创造力的图片和照片，以及相应的调色板。
- http://kuler.com：Adobe Kuler 的官方站点提供了数千种色样、调色板创建工具，以及其他人正在选用的时尚颜色。

2.11　小结

本章，我们学习了 CSS 规则，以及如何用它来为 HTML 元素应用样式。我们先是把各种 CSS 选择符都讲了一遍，灵活运用这些选择符，可以在 HTML 标记中选择目标元素。接着介绍了规则声明到底怎么把样式应用给元素的属性。在此，我们讨论了层叠、继承和特指的概念，知道了当多个样式共同影响一个属性时，哪个样式最后会胜出。最后，我们又详细罗列了在规则声明中指定属性值的各种方法，包括数值和关键字，以及可用来设定颜色的几种方法。

下一章，我们要看一看怎么设计页面文本的结构，并给它们应用样式，才能得到令人惊叹的专业布局。

3

定位元素

本章，我们该学习盒模型了。当然，还有 `position` 和 `display` 属性，以及如何浮动（float）和清除（clear）元素。这么说吧，要掌握 CSS 技术，核心就是要掌握元素定位，只有这样才能用 CSS 创造出专业水准的页面布局。

所谓盒模型，就是浏览器为页面中的每个 HTML 元素生成的矩形盒子。这些盒子们都要按照可见版式模型（visual formatting model）在页面上排布。可见的页面版式主要由三个属性控制：`position` 属性、`display` 属性和 `float` 属性。其中，`position` 属性控制页面上元素间的位置关系，`display` 属性控制元素是堆叠、并排，还是根本不在页面上出现，`float` 属性提供控制的方式，以便把元素组成成多栏布局。

3.1 理解盒模型

早在第 1 章我们就讲过了，每一个元素都会在页面上生成一个盒子。因此，HTML 页面实际上就是由一堆盒子组成的。

默认情况下，每个盒子的边框不可见，背景也是透明的，所以我们不能直接看到页面中盒子的结构。同样，我们也介绍过，使用 Web Developer 工具条，可以方便地显示出盒子的边框和背景，从而让我们能从另外一个角度来审视页面的构成。

好啦，我们先从每个元素盒子的属性开始吧。这些属性可以分成三组。

- 边框（border）。可以设置边框的宽窄、样式和颜色。
- 内边距（padding）。可以设置盒子内容区与边框的间距。
- 外边距（margin）。可以设置盒子与相邻元素的间距。



要了解有关盒模型的更多信息，请参考这里：http://www.w3.org/TR/REC-CSS2/box.html。

怎么理解这几组属性呢？最简单方法是想象一下外边距是边框向外推其他元素，而内边距是从边框向内推元素的内容，参见图 3-1。一个盒子有 4 条边，因此与边框、内边距和外边距相关的属性也各有 4 个，分别是上（top）、右（right）、下（bottom）、左（left）。

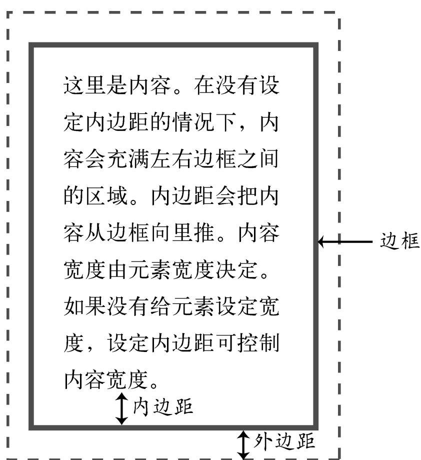


图 3-1　这个盒模型示意图展示了 HTML 元素的边框、内边距和外边距之间的关系



元素盒子还有一个背景层，可以改变颜色，也可以添加图片。与元素背景相关的属性将在本章后面介绍。

虽然可以用总共 12 个属性分别为 4 条边的边框、外边距和内边距指定宽度，但 CSS 也为我们提供了简写属性。详细信息请参考附注“简写样式”。

简写样式

CSS 为边框、内边距和外边距分别规定了简写属性，让你通过一条声明就可以完成设定。在每个简写声明中，属性值的顺序都是上、右、下、左。想象一下顺时针旋转就记住了。举个例子吧，如果要设定盒子的外边距，不用简写属性就得这样写：

```
{margin-top:5px; margin-right:10px; margin-bottom:12px; margin-left:8px;}
```

使用简写属性，则可以简写为这样：

```
{margin:5px 10px 12px 8px;}
```

注意，4 个值之间有空格，但不能是其他分隔符（比如逗号之类的）。甚至，你都不用把 4 值全都写出来——如果哪个值没有写，那就使用对边的值。

```
{margin:12px 10px 6px;}
```

对这个例子来说，由于没有写最后一个值（左边的值），所以左边就会使用右边的值，即 10px。而在下面的例子中：

```
{margin:12px 10px;}
```

只写了两个值，上和右，因此缺少的下和左就会被设定为 12px 和 10px。最后，如果你只写一个值：

```
{margin:12px;}
```

那么 4 个边都取这个值。使用这种简写属性，不能绕开上和右，只给下和左设定值，即使上和右都是零也不行。绕不开怎么办？如果它们真是零的话，那就写 0 呗，比如：

```
{margin:0 0 2px 4px;}
```

另外，每个盒子的属性也分三种粒度，到底选择哪个粒度的属性，要看你想选择哪条边，以及那条边的哪个属性。这三种粒度从一般到特殊分别是举例如下。

1. 全部 3 个属性，全部 4 条边

```
{border:2px dashed red;}
```

2. 1 个属性，全部 4 条边

```
{border-style:dashed;}
```

3. 1 个属性，1 条边

```
{border-left-style:dashed;}
```

混合使用这三种粒度的简写属性达成设计目标是很常见的。比如说吧，我想为盒子的上边和下边添加 4 像素宽的红色边框，为左边添加 1 像素宽的红色边框，而右边没有边框。可以这样写：

```
{border:4px solid red;}  /* 先给 4 条边设置相同的样式 */
{border-left-width:1px;} /* 修改左边框宽度 */
{border-right:none;}     /* 移除右边框 */
```

类似地，其他属性也都有这三级粒度，例如 padding 和 border-radius 等。

3.1.1　盒子边框

边框（border）有 3 个相关属性。

- 宽度（border-width）。可以使用 thin、medium 和 thick 等文本值，也可以使用除百分比和负值之外的任何绝对值。
- 样式（border-style）。有 none、hidden、dotted、dashed、solid、double、groove、ridge、inset 和 outset 等文本值。
- 颜色（border-color）。可以使用任意颜色值，包括 RGB、HSL、十六进制颜色值和颜色关键字。



CSS 推荐标准并未明确规定 border-width 这几个文本值 thin、medium 和 thick 的确切宽度，实际显示的宽度可能会因浏览器而异。对于边框样式（border-style），除了 solid 值（实线）之外，CSS 规范也没有明确规定。因此 dashed 值（虚线）在不同浏览器中的短划线长度和线间距也可能会不一样。



border 的第四个属性 border-radius 并不影响盒模型的定位，所以我们放在第 7 章再介绍。

前面附注栏“简写样式”里已经展示了几个边框相关的样式了，但下面我们要举几个稍微复杂点的例子，以进一步巩固你的理解。

```
p.warning {border:solid #F33;}
```

有了这条规则，任何带有 warning 类的段落都会带上一个吸引眼球的、4 像素宽的红色实心边框，如图 3-2 所示。

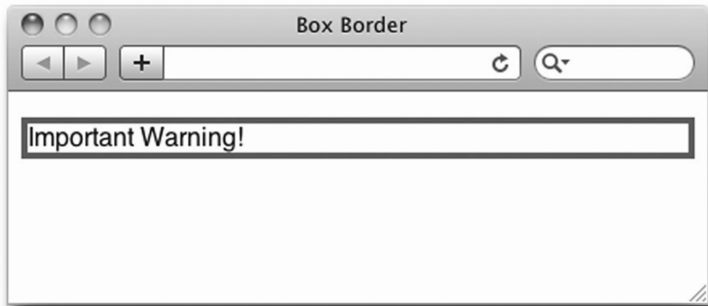


图 3-2　在这里，我希望 4 条边样式相同，因此就使用了 border 简写属性

假如我为了让边框看起来更有意思，想在 4 条边都是红色实心边框的基础上，把右边框和下边框变窄一些，那么可以在前面规则基础上新写一条规则。前面那条规则使用了“所有 4 边一次性”设定的便捷属性 border，为 4 条边设定了基础。下面这条新规则可以使用 border-width 修改个别边框的宽度：

```
p.warning {border:solid #f33;}
p.warning {border-width:4px 1px 1px 4px;}
```

结果如图 3-3 所示。

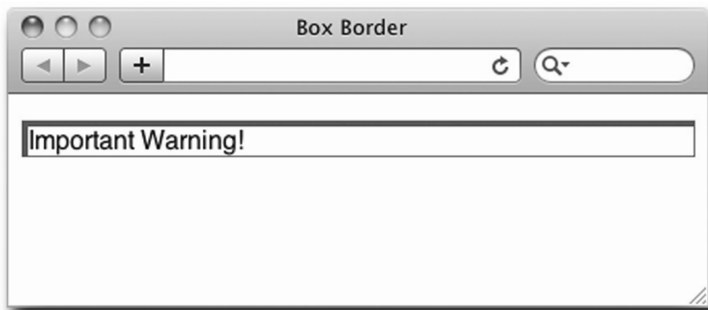


图 3-3　使用 border-width 属性，可以为盒子的每条边单独设定宽度

在实际开发的时候，为了看清楚 margin 和 padding 的实际效果，可以临时设定盒子的边框。默认情况下，边框的三个相关属性的值分别为 border-width:medium;、border-style:none;、border-color:black;。由于 border-style 的默认值是 none，所以不会显示盒子的边框。为了快速地把盒子边框显示出来，可以这样写一条规则：

```
p {border:solid 1px;}
```

这样就把 border-style 设定为实线（solid），于是边框就出现了。当然，这里也把边框宽度由默认的 3 像素变窄为 1 像素，从而把边框对布局宽度和高度的影响降到最低。

3.1.2　盒子内边距

内边距是盒子内容区与盒子边框之间的距离。图 3-4 展示了一个带边框的元素的内容区，应用给该元素的规则如下。

```
p {font:16px helvetica, arial, sans-serif; width:220px; border:2px solid red;
    background-color:#caebff;}
```

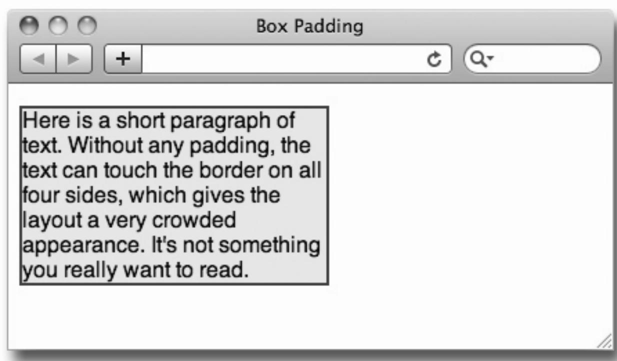


图 3-4　在没有设定内边距的情况下，内容紧挨着边框

如果你什么也不做，那么元素的文本就会像这样紧挨着元素的边框，这看着可让人有点不舒服。其实，只要给元素添加一点内边距，情况就会大为改观，如图 3-5 所示，规则如下。

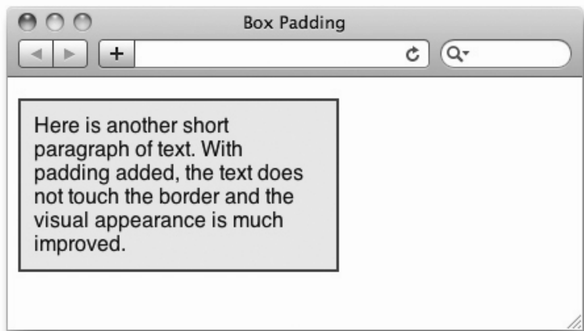


图 3-5　只要显示元素边框，为防止文本挨上它就得添加内边距

```
p {font:16px helvetica, arial, sans-serif; width:220px;border:2px solid red;
    background-color:#caebff; padding:10px;}
```

由于内边距在盒子的内部，所以它也会取得盒子的背景。仔细比较一下图 3-4 和图 3-5 就会发现，内边距实际加在了声明的盒子宽度之上。换句话说，多出来的内边距并没有像我们想象的那样挤压文本内容，这是怎么回事儿了呢？请容我先在这里卖个关子，本章后面再给大家解释。

3.1.3 盒子外边距

与边框和内边距相比，盒子的外边距要复杂一些。在图 3-6 中，有三组标题和段落。第一组标题和两个段落使用默认样式。第二组与第一组唯一的区别就是多了边框，但因此也可以看清标题与段落间的外边距有多大。第三组展示了把外边距设置为零之后的效果，该组的标题和段落全都紧挨在一起了。

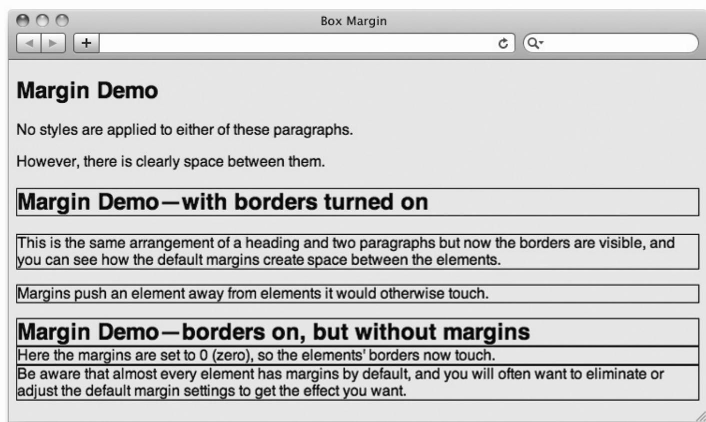


图 3–6 学会控制元素周围的外边距是布局的重要技能

中和外边距和内边距

推荐大家把下面这条规则作为样式表的第一条规则：

```
* {margin:0; padding:0;}
```

这条规则把所有元素默认的外边距和内边距都设定为零。把这条规则放到样式表里后，所有默认的外边距和内边距都会消失。然后，你可以为那些真正需要外边距的元素再添加外边距。稍后我们会介绍，不同浏览器默认的内边距和外边距也不一样，特别是对表单和列表等复合元素。

在这种情况下，用前面那条规则“中和”默认值，然后再根据需要添加，则会在各浏览器上获得一致的效果。

我在自己的项目中使用了 Eric Meyer 写的重置样式表 reset.css。这个样式表不仅重置了外边距和内边距，还对很多元素在跨浏览器显示时的外观进行了标准化。至于 Eric 为什么要写一个涉及面如此之广的重置样式表，可以参考他的文章 http://meyerweb.com/eric/thoughts/2007/04/18/reset-reasoning，reset.css 的下载地址是 http://meyerweb. com/eric/tools/css/reset。

3.1.4　叠加外边距

垂直方向上的外边距会叠加，这可是你必须得知道的一件事。本节就来解释一下什么叫外边距叠加，为什么它那么重要。假设有 3 个段落，前后相接，而且都应用以下规则：

```
/*为简明起见，省略了字体声明*/
p {height:50px; border:1px solid #000; backgroundcolor:#fff; margin-top:50px;
    margin-bottom:30px;}
```

由于第一段的下外边距与第二段的上外边距相邻，你自然会认为它们之间的外边距是 80 像素（50+30），但是你错啦！它们实际的间距是 50 像素。像这样上下外边距相遇时，它们就会相互重叠，直至一个外边距碰到另一个元素的边框。就上面的例子而言，第二段较宽的上外边距会碰到第一段的边框。也就是说，较宽的外边距决定两个元素最终离多远，没错——50 像素（参见图 3-7）。这个过程就叫外边距叠加。

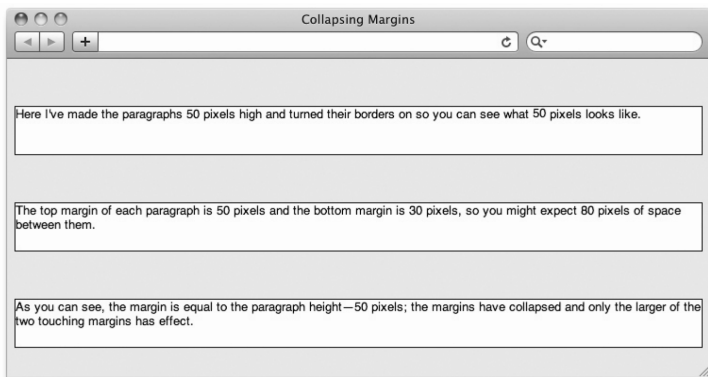


图 3-7　垂直外边距叠加（或者叫重叠），直到一个元素的外边距碰到另一个元素的边框为止



注意啦，叠加的只是垂直外边距，水平外边距不叠加。对于水平相邻的元素，它们的水平间距是相邻外边距之和。这跟你最初想的一样。

好吧，我想必须得解释一下为什么要让外边距叠加。如果有一连串段落都被应用了相同的样式，那么对其中第一段和最后一段来说，它们的上外边距和下外边距决定了它们与包含元素的间距。而那些位于中间的段落呢，根本不需要两个外边距加起来那么宽的间距。因此，就像图 3-7 所示的那样，相邻的外边距叠加起来是最合理的，哪个外边距宽，就以哪个外边距作为段间距。

3.1.5　外边距的单位

根据经验，为文本元素设置外边距时通常需要混合使用不同的单位。比如说，一个段落的左、右外边距可以使用像素，以便该段文本始终与包含元素边界保持固定间距，不受字号变大或变小的影响。而对于上、下外边距，以 em 为单位则可以让段间距随字号变化而相应增大或缩小，比如：

```
/*这里使用了简写属性把上、下外边距设置为.75em，把左、右外边距设置为 30 像素*/
p {font-size:1em; margin:.75em 30px;}
```

这样，段落的垂直间距始终会保持为字体高度的四分之三（上下外边距都是`.75em`，叠加后还是`.75em`）。如果用户增大了字号，那么不仅段落中的文本会变大，段间距也会成比例变大。这样，页面的整体布局就会比较协调一致。与此同时，使用像素单位的左、右外边距不会改变。我想，你应该也不会想让字号变化影响到布局宽度吧。

3.2　盒子有多大

无论是对初学者，还是对专家来说，W3C 盒模型的原理都是 CSS 最难理解的地方。在接下来的讨论中，我们将分别介绍块级元素（比如标题、段落和列表）和行内元素的不同行为。

为此，我们得一步一步地回顾盒模型。首先，谈一谈设定盒子的宽度，因为控制元素的宽度是创建多栏布局的头等大事。一开始我们会看到给没有宽度的元素添加边框、内边距和外边距的效果，然后再看看通过 CSS 给它设置了宽度之后，它的行为

有什么不一样。

1. 没有宽度的盒子

所谓“没有宽度”就是指没有显式地设置元素的 width 属性，这是我专门为你造的一个词——等等，求你了，真不用谢我。另外，“元素”和“盒子”从现在起代表同一个意思，至于选择哪一个，就要看到时候两者哪个最直接明了。

如果不设置块级元素的 width 属性，那么这个属性的默认值是 auto，结果会让元素的宽度扩展到与父元素同宽。下面我们来看一个宽度处于 auto 状态的元素。就使用下面这么简单的标记吧：



随着 HTML5 时代的到来，以及旧版浏览器退出历史舞台，用于确保站点在严格模式（使用 W3C 标准的盒模型）或混杂模式（适应 IE6 及更早版本的盒模型）下呈现的 XHTML 文档声明也就不再需要了。要了解什么是混杂模式，请参考这篇文章：http://www.quirksmode.org/css/quirksmode.html。

```
<body>
    <p>This element's width property is not set…</p>
</body>
```

再配上以下 CSS：

```
body {font-family:helvetica, arial, sans-serif; font-size:1em; margin:0px;
background-color:#caebff;}
p {margin:0; background-color:#fff;}
```

在包含元素 body 上，我们设定了字体、背景颜色，删除了默认的外边距，这些设定在所有盒子宽度的盒子中将一直保持不变。而对于其中的段落，我们也删除了默认的外边距。删除了外边距后，body 元素会填满浏览器窗口，而段落会填满 body 元素，如图 3-8 所示。



为了让大家看清每一步都发生了什么，我特地在窗口上方添加了一张标尺图片（代码没有给出）。这样，我也可以为了这个例子把浏览器窗口精确地缩小到 400 像素宽。

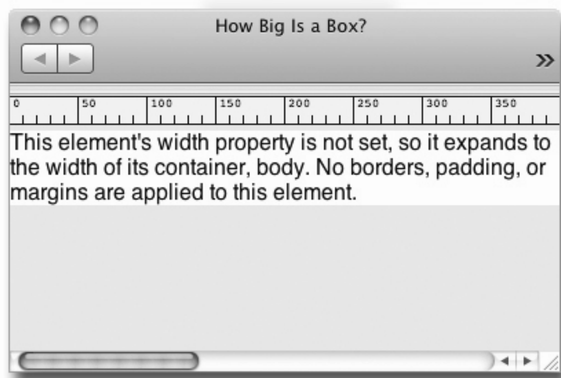


图 3-8　这个段落没有边框、内边距和外边距

在没有边框、内边距，也没有外边距的情况下，段落的文本扩展到了与 body 元素同宽。接下来用内边距给文本两侧添加一些空白，如图 3-9 所示。

```
/*为简明起见，省略了字体声明 */
p {margin:0; background-color:#fff; padding:0 20px;}
```

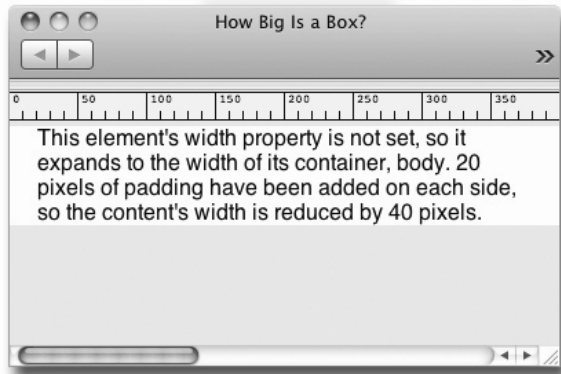


图 3-9　内边距使文本块变窄了

添加了内边距后，文本块的宽度变成了 360 像素（两边各加了 20 像素内边距）。

接下来，我们再给段落左右两边各添加 6 像素宽的边框，结果如图 3-10 所示。

```
p {margin:0; background-color:#fff; padding:0 20px;
border:solid red; border-width:0 6px 0 6px;}
```

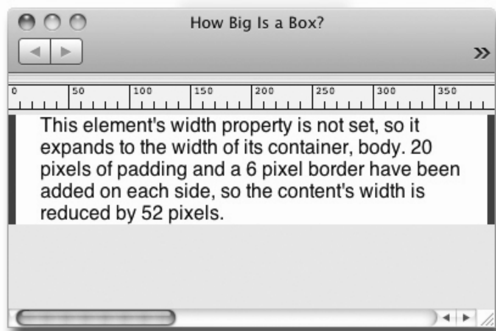


图 3-10　添加边框会进一步减少内容区的宽度

为两边各添加 6 像素的边框和 20 像素的内边距后，内容区变成了 348 像素（400-52）。最后，再给左右两边各加一些外边距，结果如图 3-11 所示。

```
p {margin:0 30px; background-color:#fff; padding:0 20px; border:solid red;
border-width:0 6px 0 6px;}
```

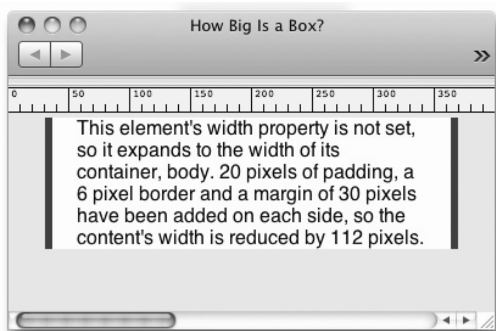


图 3-11　加上边框、内边距和外边距后，内容区变得更窄了

外边距在元素盒子与窗口之间创造了空白，此时内容宽度变成了 288 像素（$400-((20+6+30)\times 2)$）。而元素声明的总宽度并没有变，仍然是 400 像素。

盒模型结论一：没有（就是没有设置 width 的）宽度的元素始终会扩展到填满其父元素的宽度为止。添加水平边框、内边距和外边距，会导致内容宽度减少，减少量等于水平边框、内边距和外边距的和。

2. 有宽度的盒子

好了，接下来我们再做一遍同样的练习，但这一次先用 CSS 声明元素的宽度，如图 3-12 所示。

```
/*为简明起见，省略了字体声明*/
p {width:400px; background-color:#fff; margin:0;}
```

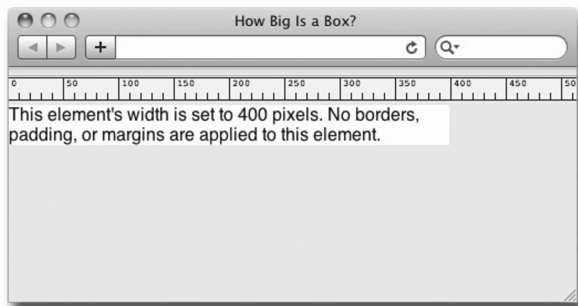


图 3-12 明确设定 width 属性后，块级元素就不会再扩展到与父元素（即 body）同宽了

这一次，段落有了固定的宽度 400 像素。在没有内边距的情况下，内容区也是声明的宽度，因此文本与盒子接触。下面给这个元素添加 20 像素的内边距：

```
p {width:400px; background-color:#fff; margin:0; padding:0 20px;}
```

有了前一个例子的经验，你可能会认为这一次内容区宽度会缩小到 360 像素，可是你错了！在给盒子设定宽度后，添加内边距会导致元素比原来宽了 40 像素，如图 3-13 所示。

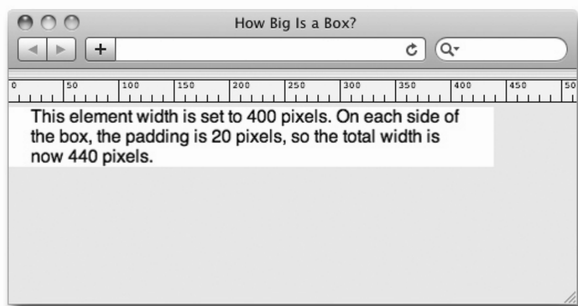


图 3-13 添加内边距使盒子变宽

要是再给盒子两边各添加 6 像素的边框呢，结果如图 3-14 所示。

```
p {width:400px; background-color:#fff; margin:0;padding:0 20px; border:solid
        red; border-width:0 6px 0 6px;}
```

盒子比刚才又宽了 12 像素。现在，原来 400 像素宽的盒子变成了 452 像素（$6 + 20 + 400 + 20 + 6 = 452$）。

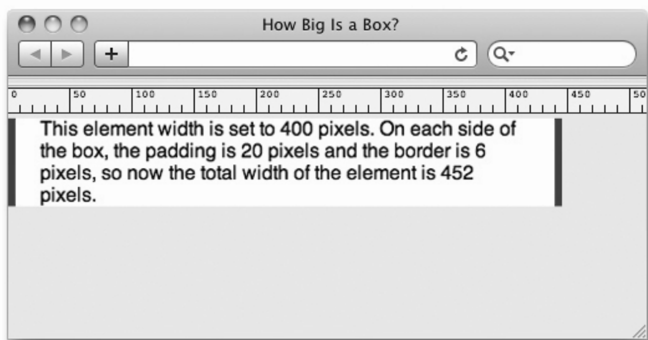


图 3–14　添加边框使盒子更宽了

最后，再给元素左、右两边添加一些外边距，如图 3-15 所示。

```
p {width:400px; background-color:#fff; margin:0 30px; padding:0 20px;
border:solid red; border-width:0 6px 0 6px;}
```

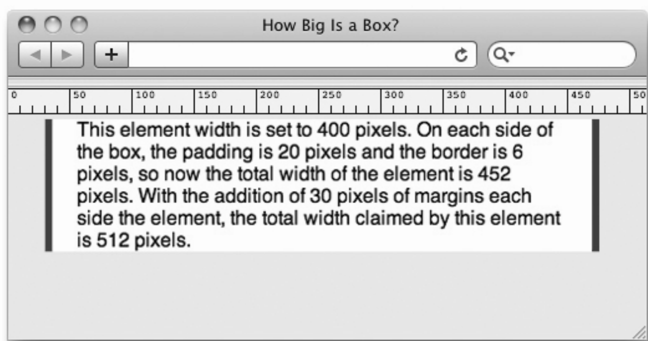


图 3–15　外边距在盒子周围添加了空白，为了展示盒子占据的空间，浏览器窗口也拉大了

添加的这 30 像素外边距，进一步增大了元素占据的空间，目前总宽度已达到 512 像素（30 + 6 + 20 + 400 + 20 + 6 + 30 = 512）。

盒模型结论二：为设定了宽度的盒子添加边框、内边距和外边距，会导致盒子扩展得更宽。实际上，盒子的 width 属性设定的只是盒子内容区的宽度，而非盒子要占据的水平宽度。

为什么我要这么不厌其烦地演示没有宽度的盒子和有宽度的盒子，在被添加了边框、内边距和外边距时所表现出来的不同行为呢？这两种盒子所表现出来的完全不同的行为，对将来构建多栏布局具有重要的启示。因为在多栏布局中，每一栏都必须时

刻维护自己的宽度。第 5 章将会讨论的“浮动布局”，在列宽由于边框、内边距和外边距被修改而意外加宽的情况下，会出现显示错误。



CSS3 新增了一个 `box-sizing` 属性，通过它可以将有宽度的盒子也设定成具有默认的 `auto` 状态下的行为。但只有最新版本的浏览器才支持该属性，所以在本书写作时（2012 年夏天），我还不能推荐你使用它。

总之，你要记住一点：设定了元素的 `width` 属性后，再给元素添加边框、内边距和外边距，元素的行为与默认的 `auto` 状态下会有截然不同的表现。

下一节我们聊一聊浮动和清除，它们可是在创建 CSS 布局之前都必须理解的关键技术。

3.3 浮动与清除

浮动和清除是用来组织页面布局的又一柄利剑，这柄剑的剑刃就是 `float` 和 `clear` 属性。浮动，你看这俩字儿多形象，意思就是把元素从常规文档流中拿出来。拿出来干什么？一是可以实现传统出版物上那种文字绕排图片的效果，二是可以让原来上下堆叠的块级元素，变成左右并列，从而实现布局中的分栏。

浮动元素脱离了常规文档流之后，原来紧跟其后的元素就会在空间允许的情况下，向上提升到与浮动元素平起平坐。

如果浮动元素后面有两个段落，而你只想让第一段与浮动元素并列（就算旁边还能放下第二段，也不想让它上来），怎么办？用 `clear` 属性来“清除”第二段，然后它就乖乖地呆在浮动元素下面了。



影响浮动的因素还有很多，推荐读者看一看 Eric Meyer 的那本 *Cascading Style Sheets 2.0 Programmer’s Reference*（2006，McGraw-Hill Osborne Media）。这里摘录 Eric 在那本书里写的一句话：“当你浮动一个元素的时候……这些（浮动）规则就好像在说：‘尽量把这个元素往上放，能放多高放多高，直到碰到某个元素的边界为止。’”即使这本书的出版年份是 2006 年，但它仍然是任何 CSS 高手必备的一本参考书。这本书涵盖了 CSS 运行机制的方方面面，其中很多在别的书里是找不到的。

在详细介绍这两个属性之前，有必要先给读者提个醒。虽然浮动元素是必要的也是非常有用的 CSS 技术，但浮动之后可能带来各种令 CSS 初学者困惑的问题。毕竟，浮动的内容已经脱离了文档流，因而无论原先在标记中包含它还是跟随它的元素，其布局都会受到它的影响。不过不要紧，本节后面展示浮动与清除的例子都经过了仔细推敲，我希望通过这些例子能让你真正理解并掌握浮动技术。

3.3.1　浮动

CSS 设计 `float` 属性的主要目的，是为了实现文本绕排图片的效果。然而，这个属性居然也成了创建多栏布局最简单的方式。



CSS3 Multi-column Layout Module 规定了如何用 CSS 定义多栏布局。但在本书写作时，只有 Opera 和 IE10 支持相应属性。因此在可以预见的未来，`float` 属性仍然是创建多栏布局的最佳途径。

那就让我们从实现文本绕排图片的效果开始吧。

1. 文本绕排图片

为了实现文本绕排图片的浮动效果，必须在标记中先写图片，然后再写环绕它的文本。

```
<img …… />
<p>…the paragraph text…</p>
```

CSS 规则如下。

```
/*为简明起见，省略了字体声明*/
p {margin:0; border:1px solid red;}
/*外边距防止图片紧挨文本*/
img {float:left; margin:0 4px 4px 0;}
```

以上规则会让图片浮动到左侧，从而让文本绕排到右侧，如图 3-16 所示。

说得形象一点，在你浮动一张图片或者其他元素时，你是在要求浏览器把它往上方推，直到它碰到父元素（也就是 `body` 元素）的内边界。后面的段落（带灰色边框）不再认为浮动元素在文档流中位于它的前面了，因而它会占据父元素左上角的位置。不过，它的内容（文本）会绕开浮动的图片。

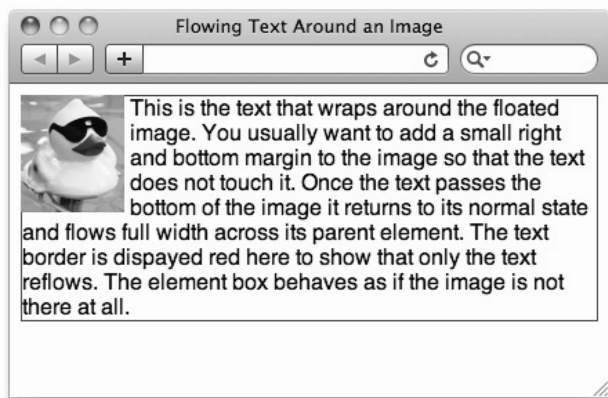


图 3–16　浮动图片会从文档流中被移除，如果在标记中有文本元素跟在它后面，则其中的文本会绕开图片



浮动非图片元素时，必须给它设定宽度，否则后果难以预料。图片无所谓，因为它本身有默认的宽度。

2. 创建分栏

在此基础上创建多栏，只要再用一次 float 属性，就这么简单。如图 3-17 所示，只要给段落设定宽度，然后也浮动它即可。

```
p {float:left; margin:0; width:200px; border:1px solid red;}
img {float:left; margin:0 4px 4px 0;}
```

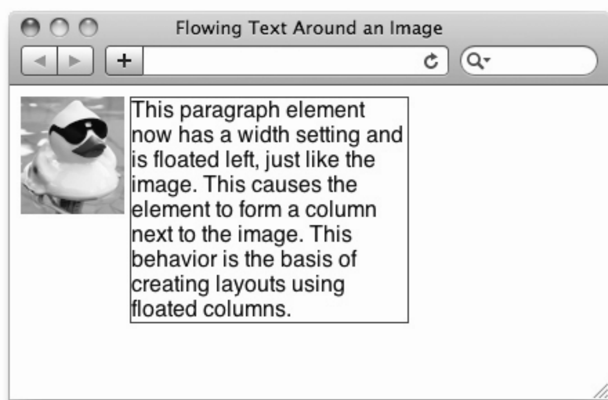


图 3–17　浮动图片旁边的固定宽度段落一经浮动，就会变成布局中的一栏，其文本也不会再绕排图片了

你这样同时浮动图片和“有宽度的”段落，会导致段落的文本绕排效果消失，而浮动的段落也会尽可能向左向上移动。就这样，这个段落就构成了紧挨着图片的一栏。这就是使用 float 属性创建多栏布局的原理。换句话说，如果几个相邻的元素都具有设定的宽度，都是浮动的，而且水平空间也足以容纳它们，它们就会并列排在一行。

如果你创建了三个浮动、固定宽度的元素，它们就会像这样并排在一行，构成三栏布局的框架。每个元素都可以作为容器，包含其他元素。关于浮动布局的内容，我们将在第 5 章再详细讨论。

接下来我们再看看浮动的另一面，这也是必须得理解的。浮动元素位于“文档流外部”，因而它已经不被包含在标记中的父元素之内了。正因为如此，它对布局可能产生破坏性影响。

3.3.2　围住浮动元素的三种方法

浮动元素脱离了文档流，其父元素也看不到它了，因而也不会包围它。这种情况有时候并非我们想要的，本节向大家传授三种围住浮动子元素的方法。记住，这三种方法你都得掌握，这样才能审时度势，选择最合适的一种。

为了演示浮动元素的行为，这种行为对布局会产生什么影响，以及解决这个问题的三种方法，我们首先要从一张带标题的图片开始。图片和标签包含在一个 section 元素中，而 section 元素后面跟着一个 footer 元素。可以把这个 footer 元素想象成很多网页底部都会有的与页面同宽的页脚。

```
<section>
    <img src="images/rubber_duck2.jpg">
    <p>It's fun to float.</p>
</section>
<footer> Here is the footer element that runs across the bottom of the
page.</footer>
```

这样，你才会知道究竟会发生什么。图 3-18 展示了应用以下规则后的 section 和 footer 的元素盒子。

```
section {border:1px solid blue; margin:0 0 10px 0;}
```

```
/*删除默认的上下外边距*/
p {margin 0;}
/*为简明起见，省略了字体声明*/
footer {border:1px solid red;}
```

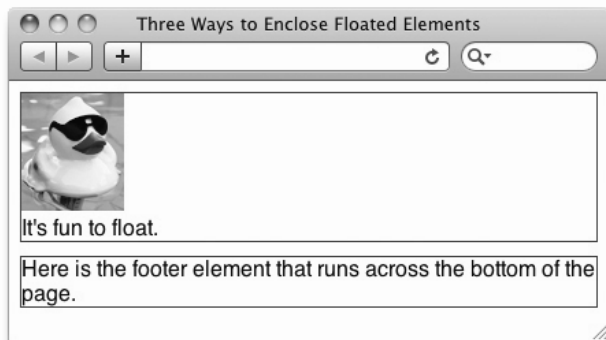


图 3-18　可以看到页面中的两个块级元素 section 和 footer，前者包含一张图片及标题，后者在常规文档流中位于前者下方

现在我们看到的是常规文档流，即块级元素包围着所有子元素，而且在页面中自上而下相互堆叠在一起。假设我们想让图片标题位于图片右侧，而不是像现在这样位于下方。运用刚刚学到的知识，我们知道实现这个目标最简单的方式就是浮动图片。试试看吧。

```
section {border:1px solid blue; margin:0 0 10px 0;}
img {float:left;}
footer {border:1px solid red;}
```

图 3-19 展示了结果。

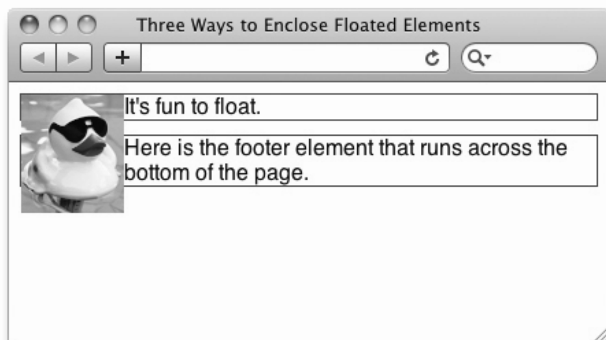


图 3-19　浮动图片后标题跑到了右边，但父元素 section 也收缩到只包含文本的高度高度

妈呀！标题倒是跑到右边了，可 section 也不再包围浮动元素了，它只包围非浮动的元素。于是，footer 被提了上来，紧挨着前一个块级元素——section。这样是没错儿，可结果呢，不是我们想要的。

方法一：为父元素添加 overflow:hidden

第一个方法很简单，缺点是不太直观，即为父元素应用 overflow:hidden，以强制它包围浮动元素。

```
section {border:1px solid blue; margin:0 0 10px 0; overflow:hidden;}
img {float:left;}
p {border:1px solid red;}
```

把 overflow:hidden 声明应用到容器元素后，footer 又回到了我们期望的位置，如图 3-20 所示。

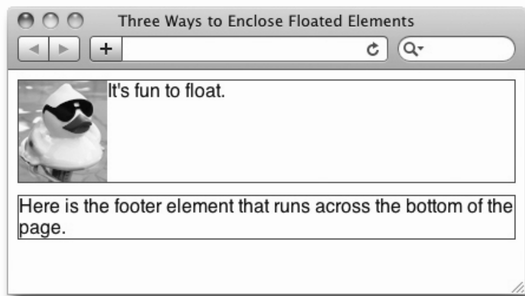


图 3-20　给容器元素应用 overflow:hidden 声明后，它又包围了浮动元素



实际上，overflow:hidden 声明的真正用途是防止包含元素被超大内容撑大。应用 overflow:hidden 之后，包含元素依然保持其设定的宽度，而超大的子内容则会被容器剪切掉。除此之外，overflow:hidden 还有另一个作用，即它能可靠地迫使父元素包含其浮动的子元素。

方法二：同时浮动父元素

第二种促使父元素包围其浮动子元素的方法，是也让父元素浮动起来。

```
section {border:1px solid blue; float:left; width:100%;}
img {float:left;}
footer {border:1px solid red; clear:left;}
```

浮动 section 以后，不管其子元素是否浮动，它都会紧紧地包围（也称收缩包裹）住

它的子元素。因此，需要用 width:100%再让 section 与浏览器容器同宽。另外，由于 section 现在也浮动了，所以 footer 会努力往上挤到它旁边去。为了强制 footer 依然呆在 section 下方，要给它应用 clear:left。被清除的元素不会被提升到浮动元素的旁边。以上代码能得到与图 3-20 相同的效果。

方法三：添加非浮动的清除元素

第三种强制父元素包含其浮动子元素的方法，就是给父元素的最后添加一个非浮动的子元素，然后清除该子元素。由于包含元素一定会包围非浮动的子元素，而且清除会让这个子元素位于（清除一侧）浮动元素的下方，因此包含元素一定会包含这个子元素——以及前面的浮动元素。在包含元素最后添加子元素作为清除元素的方式有两种。

第一种方式不太理想，也就是简单地在 HTML 标记中添加一个子元素，并给它应用 clear 属性。由于没有默认的样式，不会引入多余空间，div 元素很适合这个目的。

```
<section>
    <img src="images/rubber_duck.jpg">
    <p>It's fun to float.</p>
    <div class="clear_me"></div>
</section>
<footer> Here is the footer element…</footer>
```

在此，我为 div 添加了一个类，以便于在 CSS 中添加它。

```
section {border:1px solid blue;}
img {float:left;}
.clear_me {clear:left;}
footer {border:1px solid red;}
```

这样，浮动的元素被父元素包围住了，结果如图 3-20 所示。如果你特别不想添加这个纯表现性元素，我再告诉你一个用 CSS 来添加这个清除元素的方法。首先，要给 section 添加一个类。

```
<section class="clearfix">
    <img src="images/rubber_duck.jpg">
    <p>It's fun to float.</p>
</section>
<footer> Here is the footer element…</footer>
```

然后，再使用这个神奇的 clearfix 规则！

```
.clearfix:after {
    content:".";
    display:block;
    height:0;
    visibility:hidden;
    clear:both;
}
```



这个 clearfix 规则最早是由程序员 Tony Aslett 发明的，它只添加了一个清除的包含句点作为非浮动元素（必须得有内容，而句点是最小的内容）[①]。规则中的其他声明是为了确保这个伪元素没有高度，而且在页面上不可见。



使用 clear:both 意味着 section 中新增的子元素会清除左、右浮动元素（位于左、右浮动元素下方）。这里当然可以只用 left，但 both 也适用于将来图片 float:right 的情况。

同样，浮动的元素又像图 3-20 所示的一样被包围住了。但这次标记里没有额外硬编码的元素。好奇的话，你可以临时把 clearfix 规则中的 height 和 visibility 声明删除，看一看通过伪元素添加到标记中的句点。

我在自己写的所有网站中都使用 clearfix 规则来解决浮动问题，因为浮动是实现多栏布局（在更多浏览器支持 CSS3 的 Multi-column Layout Module 之前）唯一最可靠的方式。这一点第 5 章会跟大家详细解释。

好了，该回过头来作个总结了。要想强迫父元素包围其浮动的子元素有三种方式，哪三种？

- 为父元素应用 overflow:hidden
- 浮动父元素
- 在父元素内容的末尾添加非浮动元素，可以直接在标记中加，也可以通过给父元素添加 clearfix 类来加（当然，样式表中得需要相应的 clearfix 规则）

这三种方法的使用要因地制宜。比如，不能在下拉菜单的顶级元素上应用 overflow:hidden，否则作为其子元素的下拉菜单就不会显示了。因为下拉菜单会显示在其父元素区域的外部，而这恰恰是 overflow:hidden 所要阻止的。再比如，不能对已经靠自动外边距居中的元素使用“浮动父元素”技术，否则它就不会再居中，

① after 会在元素内容后面而不是元素后面插入一个伪元素。——译者注

而是根据浮动值浮动到左边或右边了。总之，掌握了这三种技术之后，再碰到需要包围浮动元素的情况，你就能够游刃有余了。

没有父元素时如何清除

有时候，在清除某些浮动元素时，不一定正好有那么个父元素可以作为容器来强行包围它们。此时，最简单的办法就是给一个浮动元素应用 `clear:both`，强迫它定位在前一个浮动元素下方。然而，在空间足以容纳多个元素向上浮动时，这个简单的办法未必奏效，我们还得另辟蹊径。

为了演示这种情况，图 3-21 展示了一个页面，其中包含 6 个元素：3 张图片和介绍它们的 3 个文本段落。这个页面布局是通过浮动图片实现的，因此在标记中跟在图片后面的文本会向上走，停靠在浮动图片的右侧。

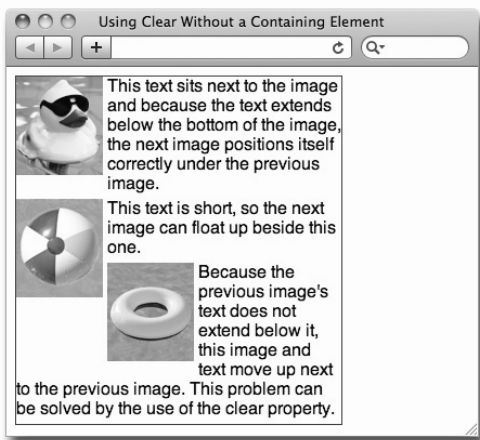


图 3-21　由于第二段文字下方有空间，所以第三张图片及说明文字会上浮到第二张图片右侧，这不是我们想要的结果

以下是图 3-21 中页面对应的 HTML（为节省版面，删除了部分文字）：

```
<section>
    <img src="images/rubber_duck3.jpg">
    <p>This text sits next to the image and because the…</p>
    <img src="images/beach_ball.jpg">
    <p>This text is short, so the next image can float up…</p>
    <img src="images/yellow_float.jpg">
    <p>Because the previous image’s text does not…</p>
</section>
```

相应的 CSS 如下：

```
section {width:300px; border:1px solid red;}
img {float:left; margin:0 4px 4px 0;}
/*为简明起见，省略了字体声明*/
p {margin:0 0 5px 0;}
```

我们的目标是让每段文字停靠在相应的图片旁边。然而，第二段文字太短了，都没有够到第二张浮动图片的下沿。这就给下一对儿图片/段落向上浮动留出了空间。

这个例子中的布局效果从技术角度看是正确的：第三对儿图片/段落有条件（有空间）停靠在第二张浮动图片旁边，于是它们就毫不客气地靠了过去。毕竟，浮动元素的使命不就是尽可能地向左上或右上迁移吗。可这个视觉上的结果却不是我们想要的。

由于每一对儿图片/段落都没有包含元素，在此就无法使用前面讨论的“强制父元素包围”的战术。不过，我照样可以使用 clearfix 规则呀！

```
.clearfix:after {
    content:".";
    display:block;
    height:0;
    visibility:hidden;
    clear:both;
}
```

像这样给每个段落都加上 clearfix 类：

```
<section>
    <img src="images/rubber_duck3.jpg">
    <p class="clearfix">This text sits next to the image and because the…</p>
    <img src="images/beach_ball.jpg">
    <p class="clearfix">This text is short, so the next image can float up…</p>
    <img src="images/yellow_float.jpg">
    <p class="clearfix">Because the previous image's text does not…</p>.
</section>
```

如图 3-22 所示，在每个段落内容的最后添加了“清除子元素”，我们想要的布局效果实现了。因为第三对儿图片和段落前面增加了一个清除元素，所以它们就不能再往上走了。注意，我没有只给第二个段落添加 clearfix 类，而是每个段落都加上了一

个。如果是真正的网站开发，就得这么做啊。这样，无论将来哪个段落的文本高度低于图片了，页面布局都不会被破坏。

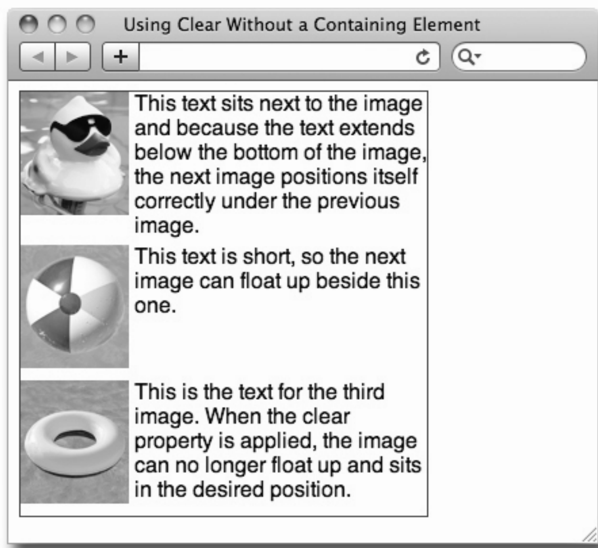


图 3-22　通过 clearfix 类添加了清除元素后，布局就跟我们期望的一样了

说到这儿，相信读者对 float 和 clear 属性都有了深入理解了。本章最后，再给大家介绍两个对 CSS 布局至关重要的属性：position 和 display。

3.4　定位

CSS 布局的核心是 position 属性，对元素盒子应用这个属性，可以相对于它在常规文档流中的位置重新定位。position 属性有 4 个值：static、relative、absolute、fixed，默认值为 static。这些属性都是什么意思？别急，我会通过以下 4 个段落来逐个说明。

```
<p>First Paragraph</p>
<p>Second Paragraph</p>
<p id="specialpara">Third Paragraph (with ID)</p>
<p>Fourth Paragraph</p>
```

在接下来的例子中，我会让第一段、第二段和第四段保持默认的 static 定位方式，但修改第三段的 position 属性。

为了不影响其他段落，我特意为第三段添加了值为 specialpara 的 ID。

3.4.1 静态定位

我们先看一看图 3-23，这是四个段落都采用默认静态定位的效果。

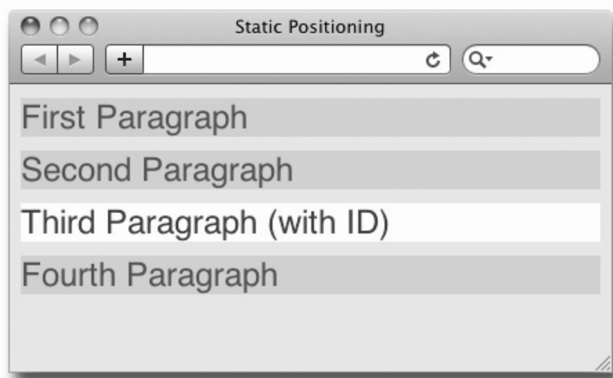


图 3-23 静态定位下的块级元素会在默认文档流中上下堆叠

在静态定位的情况下，每个元素在处在常规文档流中。它们都是块级元素，所以就会在页面中自上而下地堆叠起来。想想第 1 章我们展示的那个没有应用样式的 HTML 布局，那就是默认的 static 文档流。

要想突破 static 定位提供的这种按顺序布局元素的方式，必须把盒子的 position 属性改为其他三个值。

3.4.2 相对定位

下面我们就把第三段的 position 属性设置为 relative。光设置这个属性还看不出来有什么不一样，因为你只设置了它的定位方式是“相对定位”。到底相对哪里定位呢？相对的是它原来在文档流中的位置（或者默认位置）。接下来，可以使用 top、right、bottom 和 left 属性来改变它的位置了。但多数情况下，只用 top 和 left 就可以实现我们想要的效果。以下 CSS 规则

```
p#specialpara {position:relative; top:25px; left:30px;}
```

可以得到图 3-24 所示的结果。

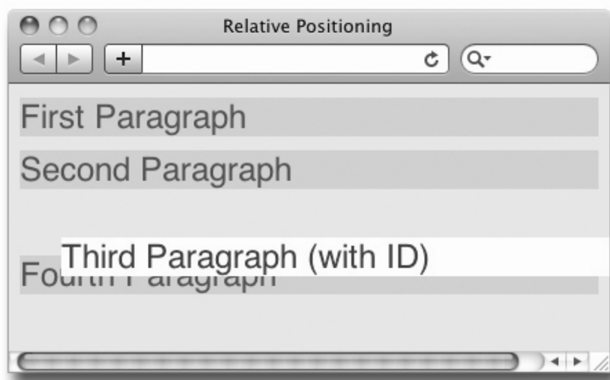


图 3-24　相对定位下，可以利用 top 和 left 属性相对于元素在文档流中的常规位置重新定位



可以给 top 和 left 属性设定负值，把元素向上、向左移动。

现在，第三段与它在文档流中的默认位置相比，向下移动了 25 像素，向右移动了 30 像素。相当于它把自己从原来的包含元素（body）中挣脱出来了，而且有一部分还跑到了屏幕之外。要注意，除了这个元素自己相对于原始位置挪动了一下之外，页面没有发生任何变化。换句话说，这个元素原来占据的空间没有动，其他元素也没动。

使用相对定位的关键是什么呢？就是要考虑到元素原来的空间。如图 3-24 所示，可以给第四段设置一个 30 像素或更大的 margin-top 值，让它向下移动，从而避免被重新定位的第三段挡住。

3.4.3　绝对定位

绝对定位跟静态定位和相对定位比，绝对不一样。因为绝对定位会把元素彻底从文档流中拿出来。好，下面我们就修改例子中的代码，把 relative 改成 absolute。

```
p#specialpara {position:absolute; top:25px; left:30px;}
```

图 3-25 显示了结果。

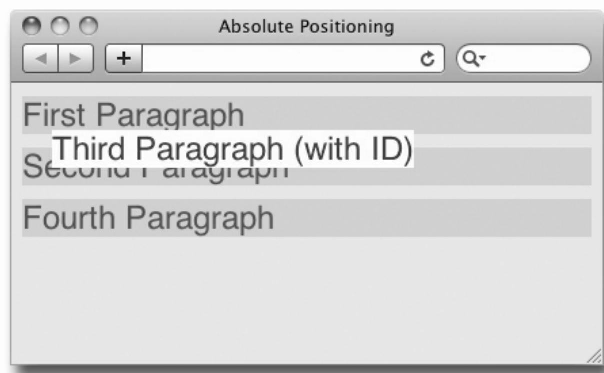


图 3-25　绝对定位下，元素从文档流中被“连根拔起”，然后再相对于其他元素（在这里，是默认的定位上下文 body）定位

在图 3-25 中，可以看到元素之前占据的空间被“回收了”。这说明，绝对定位的元素完全脱离了常规文档流，它现在是相对于顶级元素 body 在定位。而这自然而然就引出了一个关于定位的重要概念：定位上下文。

关于定位上下文，首先我们要知道绝对定位元素默认的定位上下文是 body 元素。如图 3-25 所示，通过 top 和 left 设定的偏移值，决定了元素相对于 body 元素（标记层次中的祖先容器），而不是相对于它在文档流中的位置偏移多远——这一点与相对定位的元素不同。

由于绝对定位元素的定位上下文是 body，所以在页面滚动的时候，为了维护与 body 元素的相对位置关系，它也会相应地移动。

在介绍怎么给绝对定位元素设定其他定位上下文（body 之外的元素）之前，我们先把 4 种定位方式介绍完。接下来看一看固定定位。

3.4.4　固定定位

从完全移出文档流的角度说，固定定位与绝对定位类似。

```
p#specialpara {position:fixed; top:30px; left:20px;}
```

但不同之处在于，固定定位元素的定位上下文是视口（浏览器窗口或手持设备的屏幕），因此它不会随页面滚动而移动。图 3-26 和图 3-27 展示了固定定位的效果。

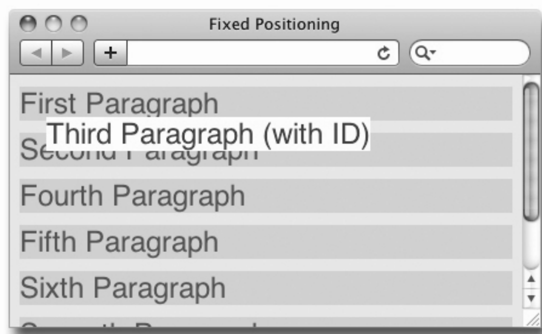


图 3-26 乍一看，固定定位很像绝对定位……

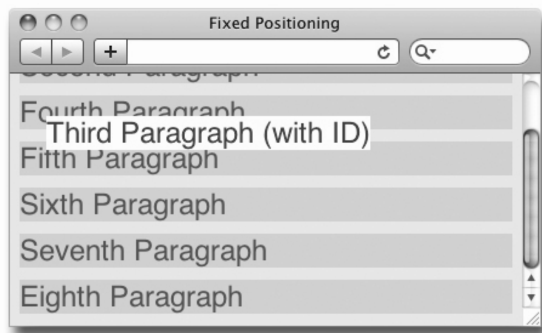


图 3-27 ……但滚动页面才发现，固定定位元素不随着页面滚动而移动

固定定位并不常用，最常见的情况是用它创建不随页面滚动而移动的导航元素。

好了，既然你已经理解了 4 个定位属性，那接下来我们好好讲一讲定位上下文吧。

3.4.5 定位上下文

把元素的 `position` 属性设定为 `relative`、`absolute` 或 `fixed` 后，继而可以使用 `top`、`right`、`bottom` 和 `left` 属性，相对于另一个元素移动该元素的位置。这里的“另一个元素”，就是该元素的定位上下文。

在讲绝对定位的时候，我们知道绝对定位元素默认的定位上下文是 `body`。这是因为 `body` 是标记中所有元素唯一的祖先元素。而实际上，绝对定位元素的任何祖先元素都可以成为它的定位上下文，只要你把相应祖先元素的 `position` 设定为 `relative` 即可。

比如下面的标记

```
<body>
    <div id="outer">
        <div id="inner">This is text…</div>
    </div>
</body>
```



请注意，对 HTML 中的文本应该使用恰当的语义标签来标记。我们这里为了说明问题的需要，才把文本直接放在了没有语义的 div 中。

搭配下面的 CSS

```
div#outer {width:250px; margin:50px 40px; border-top:3px solid red;}
div#inner {top:10px; left:20px; background:#ccc;}
```

结果如图 3-28 所示。

看到代码里给内部 div 设定了 top 和 left 属性，你是不是觉得图 3-28 有问题——为什么内部 div 没有相对外部 div 向下移动 10 像素，向右移动 20 像素呢？为什么它们俩的原点（左上角点）还一样呢？原因在于，内外部 div 默认都是静态定位，它们之间不存在谁是谁的定位上下文这个问题。换句话说，在常规文档流中，由于外部 div 没有内容，内部 div 就会跟它共享相同的起点。只有将元素的 position 属性设定为 relative、absolute 或 fixed，这个元素的 top、right、bottom 和 left 属性才会起作用。下面我们就把内部 div 设定为绝对定位，来看一看有什么变化发生。

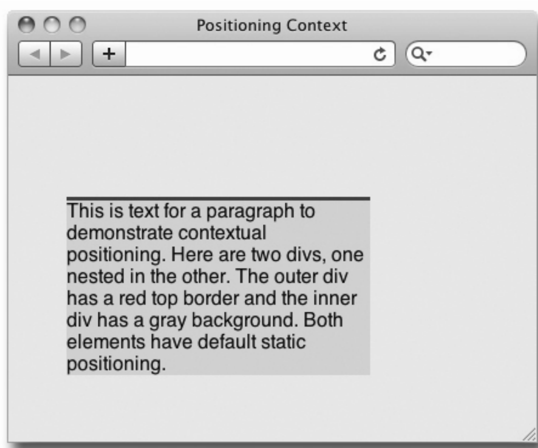


图 3-28　这里是两个嵌套在一起的 div。我们给外部 div 的上方加了边框，给内部 div 加了背景。由于内部 div（默认）是静态定位的，因此 top 和 left 属性不起作用

```
div#outer {width:250px; margin:50px 40px; border-top:3px solid red;}
div#inner {position:absolute; top:10px; left:20px; background:#ccc;}
```

对了，绝对定位相对于谁呀？由于没有相对定位的祖先元素供其参照，内部 div 只能以默认的定位上下文 body 作为参照，相对于它定位。此时，内部 div 完全无视其父元素（外部 div）的存在，top 和 left 属性会相对于 body 元素向下、向左偏移其位置，结果如图 3-29 所示。

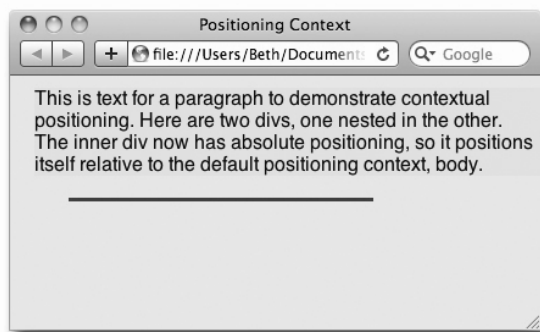


图 3-29 虽然有背景的内部 div 在标记中位于外部 div（看那个边框）之中，但由于不存在相对定位的其他祖先元素可以作为定位上下文，绝对定位的内部 div 只能相对于 body 元素进行定位



事实上，只要把元素的外边距和内边距设定好，多数情况下只用静态定位就足以实现页面布局了。很多刚开始接触 CSS 的初学者都会错误地设定 position 属性，最终才发现从文档流中挪出来的这些元素一点也不好控制。因此，除非真需要那么做，否则不要轻易修改元素默认的 position 属性。

如果我现在把外部 div 的 position 属性设定为 relative：

```
div#outer {position:relative; width:250px; margin:50px 40px; border-top:3px
    solid red;}
div#inner {position:absolute; top:10px; left:20px; background:#ccc;}
```

这样，绝对定位的内部 div 的定位上下文就变成了外部 div，结果如图 3-30 所示。

此时内部 div 的 top 和 left 属性参照的就是外部 div 了。如果你再用 left 和 top 属性重新定位外部 div，内部 div 也会跟着移动相同的距离，以保证它与外部 div（也就是它的定位上下文）之间的位置关系。

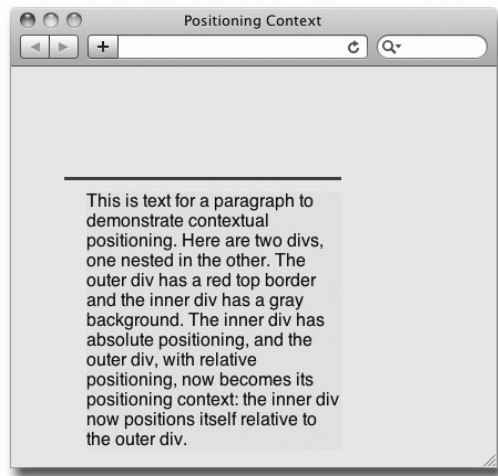


图 3-30　外部 div 改为相对定位之后，其后代中绝对定位的元素就会按照 top 和 left 属性的设定，相对于外部 div 定位

3.5　显示属性

正如所有元素都有 position 属性，所有元素也都有 display 属性。尽管 display 属性的值有很多，但大多数元素 display 属性的默认值不是 block，就是 inline。要是你在我们讲第 1 章的时候睡着了，我单独给你讲一讲块级元素与行内元素的区别。

- 块级元素，比如段落、标题、列表等，在浏览器中上下堆叠显示。
- 行内元素，比如 a、span 和 img，在浏览器中左右并排显示，只有前一行没有空间时才会显示到下一行。

把块级元素变成行内元素（或者相反）的魔法如下。

```
/*默认为 block*/
p {display:inline;}
/*默认为 inline*/
a {display:block;}
```

这种转换可以让原先的行内元素填满其父元素。本书后面在介绍到 CSS 下拉菜单的时候，就会用到这个技巧。

display 属性还有一个值有必要提一下，就是 none。把元素的 display 设定为 none，该元素及所有包含在其中的元素，都不会在页面中显示。它们原先占据的所有空间

也都会被“回收”，就好像相关的标记根本不存在一样。与此相对的是 `visibility` 属性，这个属性最常用的两个相对的值是 `visible`（默认值）和 `hidden`。把元素的 `visibility` 设定为 `hidden`，元素会隐藏，但它占据的页面空间仍然“虚位以待”。

3.6 背景

本章的主题是定位元素，关于定位元素，最后要讲的一个主题是背景。背景支持为元素添加背景颜色和背景图片。要是你使用过 Adobe Photoshop 或 Adobe Fireworks 的话，一定知道图层这个概念。CSS 里也一样，每个元素盒子都可以想象成由两个图层组成。元素的前景层包含内容（如文本或图片）和边框，元素的背景层可以用实色填充（使用 `background-color` 属性），也可以包含任意多个背景图片（使用 `background-image` 属性），背景图片叠加在背景颜色之上。

在 CSS3 被浏览器实现之前，只能在背景颜色上添加一张背景图片。而现在，我们可以为背景图层添加多张图片（以及 CSS3 渐变）。为了让大家对元素盒子的图层有更直观的认识，下面我们把本章开头的盒模型示意图改成三维透视图，如图 3-31 所示。

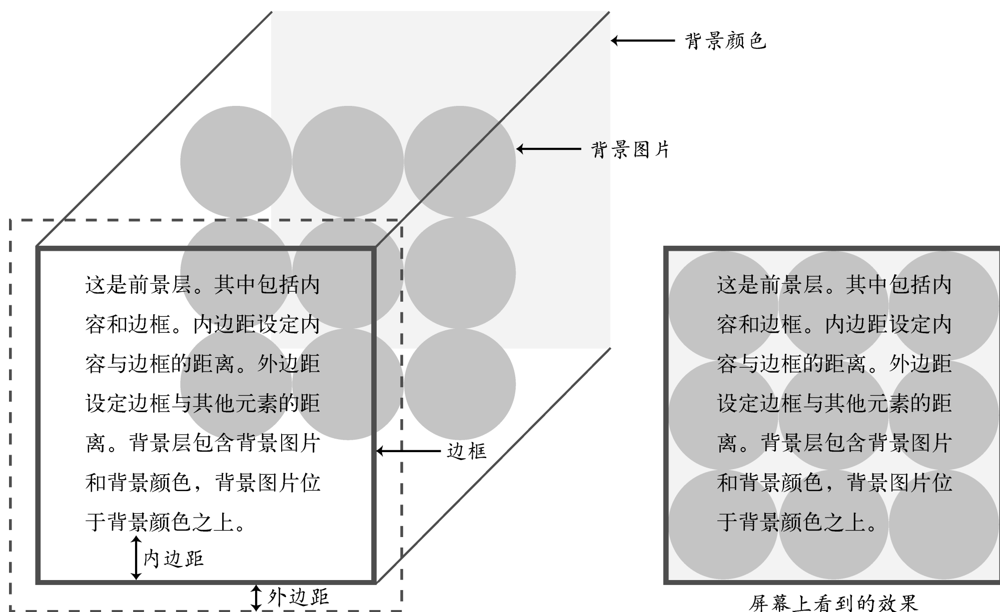


图 3-31　这个盒模型示意图展示了元素的前景和背景层

3.6.1　CSS背景属性

CSS 规定以下与背景相关属性。

- ❑ background-color
- ❑ background-image
- ❑ background-repeat
- ❑ background-position
- ❑ background-size
- ❑ background-attachment
- ❑ background（简写属性）
- ❑ background-clip、background-origin、background-break（目前尚未得到广泛支持）

这些属性可以让我们控制背景图层的方方面面。下面我们就来一个一个地讲解。

3.6.2　背景颜色

background-color 是背景属性中最简单的，通过它可以设定元素的颜色。然后，元素就会以设定的颜色填充背景图层，如图 3-32 所示。

```
body {background-color:#caebff;}
p {/*盒子布局样式*/
    font-family:helvetica, arial, sans-serif; font-size:18px;
    width:350px; margin:20px auto; padding:10px;
    /*这个例子中讨论背景和前景样式*/
    background-color:#fff; color:#666; border:4px solid;
}
```

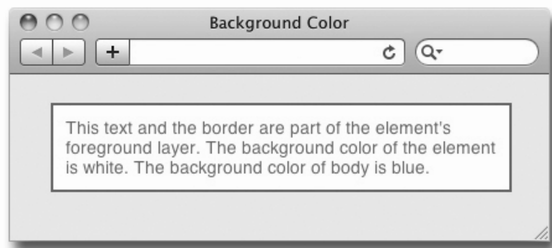


图 3-32　body 的 background-color 是蓝绿色，段落的 background-color 是白色，前景色 color 是灰色，前景色既影响文本，也影响边框

这个例子除了演示怎么给元素添加背景色，还演示了前景色的作用范围，也就是前景色会影响元素的内容和边框。当然，有一个前提条件，就是在使用 border 设定边框的样式和宽度，而没有设定边框颜色（或者没有使用 border-color 单独设定边框颜色）的情况下，边框会使用 color 属性设定的字体颜色。默认颜色是黑色。如果你想让边框的颜色有别于文本，就需要单独设定。

3.6.3 背景图片

图 3-33 是一张包含圆形图案的图片，本节我们就它来示范 background-image 和 background-repeat 属性。

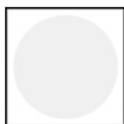


图 3-33 为包含圆形图案的图片添加了边框，圆形四周有空白

接下来，我们通过 background-image 属性把这张包含圆形图案的图片放到元素的背景层上，结果如图 3-34 所示。

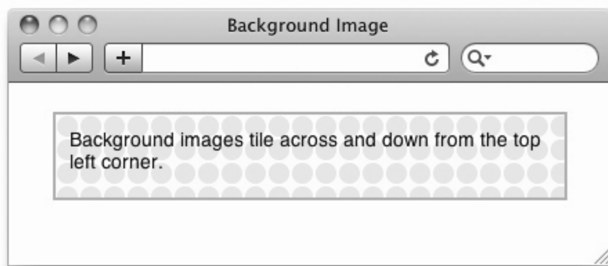


图 3-34 比元素小的背景图片会在水平和垂直方向上重复出现，直至填满整个背景空间

```
p {
    font-size:28px;
    font-family:helvetica, arial, sans-serif;
    width:345px;
    height:110px;
    margin:20px auto;
    padding:10px;
    color:#000;
    border:4px solid #aaa;
```

```
background-color:#fff;
background-image:url(images/blue_circle.png);}
```

由此可见，默认情况下背景图片会以元素左上角为起点，沿水平和垂直方向重复出现，最终填满整个背景区域。正是因为以元素左上角为原点，所以元素盒子底部和右侧的圆形图案都只显示了一部分。要注意的是，指定背景图片来源的方式，与 img 标签中的方式不同，要这样：

```
background-image:url(图片路径/图片文件名)
```

图片地址两边不用加引号，当然加了也没问题。

要改变默认的水平和垂直重复效果，可以修改 background-repeat 属性；要改变背景图片的起点，可以修改 background-position 属性。

3.6.4　背景重复

控制背景重复方式的 background-repeat 属性有 4 个值。默认值就是 repeat，效果就是图 3-34 中所示的水平和垂直方向都重复，直至填满元素的背景区域为止。另外 3 个值分别是只在水平方向重复的 repeat-x、只在垂直方向上重复的 repeat-y 和在任何方向上都不重复（或者说只让背景图片显示一次）的 no-repeat 。这几个值的效果如图 3-35 所示。

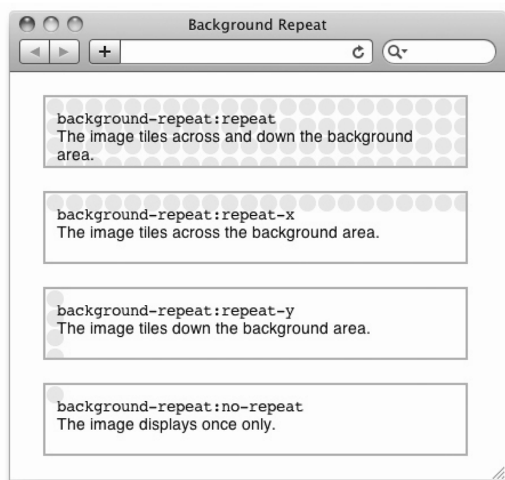


图 3–35　4 个不同 background-repeat 值的效果

这几种重复方式的用法有很多种。比如，repeat-x 和 repeat-y 可以用来实现装饰性的边框效果，而 no-repeat 则控制背景图片只出现一次。除了背景图片、背景重复之外，另一个相关的控制选项是 background-position，我们放在下一节介绍。

在此之前先提醒大家一句，CSS3 还规定另外两个值（但尚未得到浏览器支持），以控制背景图片重复确切的次数，即所有图片都是完整的，不会出现半张图片的现象。

- background-repeat:round：为确保图片不被剪切，通过调整图片大小来适应背景区域。
- background-repeat:space，为确保图片不被剪切，通过在图片间添加空白来适应背景区域。

3.6.5 背景位置

用于控制背景位置的 background-position 属性，是所有背景属性中最复杂的。background-position 属性有 5 个关键字值，分别是 top、left、bottom、right 和 center，这些关键字中的任意两个组合起来都可以作为该属性的值。比如，top right 表示把图片放在元素的右上角位置，center center 把图片放在元素的中心位置。事实上，这都是很含糊的说法，下面我们就来详细解释。

千万要注意，background-position 属性同时设定元素和图片的原点。原点决定了元素和图片中某一点的水平和垂直坐标。默认情况下，background-position 的原点位于左上角。换句话说，元素的左上角和图片的左上角是对齐的，随后图片向各个方向重复，都是以左上角为起点。图 3-35 中所示就是默认以左上角为原点的情形。

有了这个基本共识之后，下面我们就在实践中学习 background-position 属性，仍以图 3-35 中的第一种情况为例。第一情况下，background-position 属性的默认值 top left 控制着水平和垂直方向重复的起点。那我们看一看，把起点位置改为 center center 之后会有什么不一样（结果参见图 3-36）。

```
/*center center 的简化写法*/
p#center {background-position:center;}
```



只给 background-position 设定一个关键字值，则另一个也会取相同的值（比如这里就相当于写了 background-position:center cetner）。

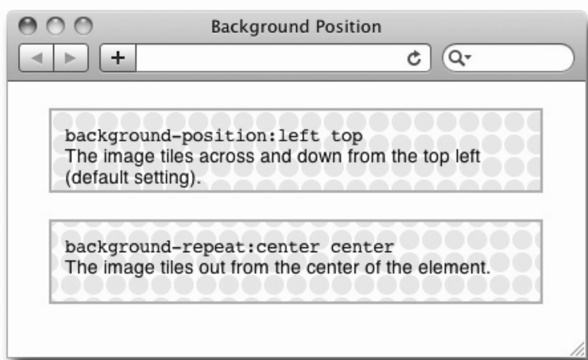


图 3-36　background-position:center center 设定图片中心点与元素中心点重合，然后再向各个方向重复

比较修改前后的结果会发现，第二段中的背景图片是以段落的中心点为起点，然后再向水平和垂直方向重复。

好了，换一种思路。这次我们用百分比来设定位置，结果如图 3-37 所示。

```
div {
    height:150px;
    width:250px;
    border:2px solid #aaa;
    margin:20px auto;
    background-image:url(images/turq_spiral_150.png);
    background-repeat:no-repeat;
    background-position:50% 50%;
}
```

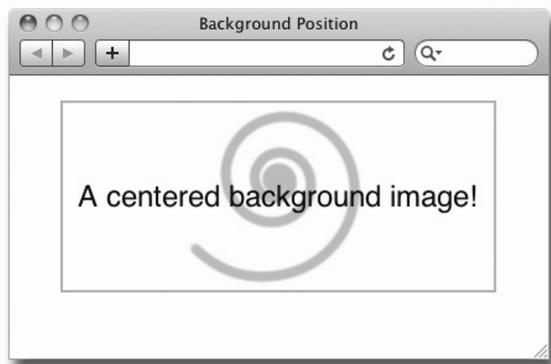


图 3-37　使用 background-position 把背景图片居中



我听见有人小声嘀咕："为什么文本也跟着垂直居中了呢？"哈哈，这是因为我把 line-height 设定成了元素的高度，而行高是在文本行上下平均分配的。此外，我还把 text-align 设定为 center，把文本水平居中。这样就让文本跟背景图片一样，在两个方向上都居中了。

通过把 background-position 设定为 50% 50%，把 background-repeat 设定为 no-repeat，实现了图片在背景区域内居中的效果。

背景位置的值

设定背景位置时可以使用三种值：关键字、百分比、绝对或相对单位的数值。可以使用两个值分别设定水平和垂直位置。

关键字指的顺序不重要，left bottom 和 bottom left 意思相同。为了设定的值在所有浏览器中都有效，最好不要混用关键字值与数字值。

使用数值（比如 40% 30%）时，第一个值表示水平位置，第二个值表示垂直位置。要是只设定一个值，则将其用来设定水平位置，而垂直位置会被设为 center。

在使用关键字和百分比值的情况下，设定的值同时应用于元素和图片。换句话说，如果设定了 33% 33%，则图片水平 33%的位置与元素水平 33%的位置对齐。垂直方面也一样。图 3-37 所示也是一个例子，那是通过 center center 把图片的中心点定位在了元素的中心点。

像素之类的绝对单位数值就不一样了。要是用像素单位来设定位置，那么图片的左上角会被放在距离元素左上角指定位置的地方。

有意思的是，还可以使用负值。这样就可以把图片的左上角定位到元素外部，从而在元素中只能看到部分图片。当然，给图片设定足够大的正值，也可以把图片的右下角推到元素外部，从而在元素中也只能看到部分图片。位于元素外部的那部分图片不会显示。

3.6.6 背景尺寸

background-size 是 CSS3 规定的属性，但却得到了浏览器很好的支持。这个属性用来控制背景图片的尺寸，可以给它设定的值及含义如下。

- 50%：缩放图片，使其填充背景区的一半。

- 100px 50px：把图片调整到 100 像素宽，50 像素高。
- cover：拉大图片，使其完全填满背景区；保持宽高比。
- contain：缩放图片，使其恰好适合背景区；保持宽高比。

仍然使用图 3-37 居中背景图片的 CSS 规则，但把图片换一换，再分别设定上面列出的 background-size 属性的几个值，会得到图 3-38 所示的效果。

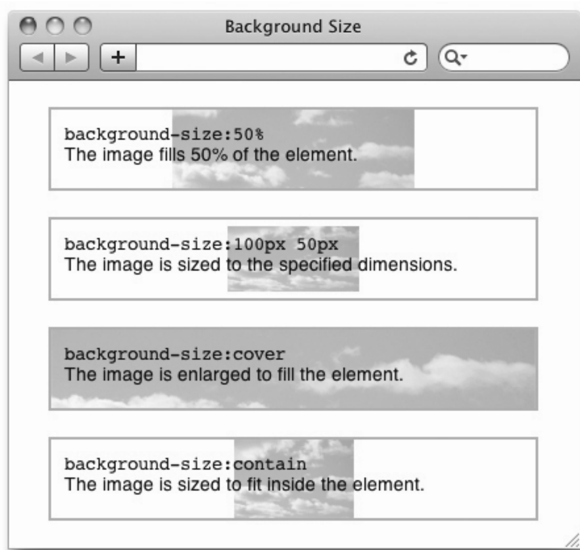


图 3–38　给一个居中的不重复的背景图片应用不同的 background-size 值的效果

这个新属性为我们控制背景图片提供了更多可能性。使用这个属性需要注意，别把本来很小的图片拉得太大，那样会导致图片质量失真。

3.6.7　背景粘附

background-attachment 属性控制滚动元素内的背景图片是否随元素滚动而移动。这个属性的默认值是 scroll，即背景图片随元素移动。如果把它的值改为 fixed，那么背景图片不会随元素滚动而移动。

background-attachment:fixed 最常用于给 body 元素中心位置添加淡色水印，让水印不随页面滚动而移动。

实现这种效果的 CSS 规则如下。

```
body {
    background-image:url(images/watermark.png);
    background-position:center;
    background-color:#fff;
    background-repeat:no-repeat;
    background-size:contain;
    background-attachment:fixed;
}
```

没错，关于背景图片的规则写起来有点费劲，因为属性名太长了。别担心，使用简写属性 background 就可以在一条声明里设置所有值。

3.6.8 简写背景属性

background 属性可以用来设定所有背景相关的值。比如，前面那个 background-attachment 的例子使用简写的 background 属性，可以写成这样一条规则：

```
body {background:url(images/watermark.png) center #fff no-repeat contain fixed;}
```

声明中少写了哪个属性的值（比如没写 no-repeat），就会使用相应属性的默认值（repeat）。

3.6.9 其他CSS3 背景属性

CSS3 又新增了一些新的背景属性，这里来简单介绍一下。不过，这些属性受支持的程度并不一致，如果你想使用它们，别忘了测试自己的页面在这些属性不可用时会出什么问题。要不，就使用 Modernizr 来检测浏览器对它们的支持情况，并为不支持它们的浏览器提供替代 CSS。



Modernizr 是一个 JavaScript 库，用于检测用户浏览器支持哪些 HTML5 和 CSS3 功能。更多信息，请参考这个网址：http://modernizr.com。

- background-clip。控制背景绘制区域的范围，比如可以让背景颜色和背景图片只出现在内容区，而不出现在内边距区域。默认情况下，背景绘制区域是扩展到边框外边界的。

- background-origin。控制背景定位区域的原点，可以设定为元素盒子左上角以外的位置。比如，可以设定以内容区左上角作为原点。
- background-break。 控制分离元素（比如跨越多行的行内盒子）的显示效果。



有关这些新属性的更多信息，请参考：http://www.w3.org/TR/css3-background。

3.6.10　多背景图片

CSS3 还可以给元素背景添加多个背景图片，下面我们就使用简写属性 background 来说明，效果见图 3-39。

```
p {
    height:150px;
    width:348px;
    border:2px solid #aaa;

    margin:20px auto;
    font:24px/150px helvetica, arial, sansserif;

    text-align:center;
    background:
    url(images/turq_spiral.png) 30px -10px no-repeat,
    url(images/pink_spiral.png) 145px 0px no-repeat,
    url(images/gray_spiral.png) 140px -30px no-repeat, #ffbd75;
}
```



图 3–39　多张图片可以在背景中叠加起来，CSS 规则中先列出的图片在上层

在 CSS 中，我把每张图片的声明都单独放在了一行里，以逗号分隔，以便看清它们的位置、重复的设定值。为了防止图片加载失败时元素背景处于默认的透明状态，这里也在最后一条声明中加上了背景颜色（加粗的值）。要注意的是，代码中先列出的图片显示在上方，或者说，更接近前景。

厂商前缀

为鼓励浏览器厂商尽早采用 W3C 的 CSS3 推荐标准，于是就产生了 VSP（Vendor Specific Prefixes，厂商前缀）的概念。

有了这些 CSS 属性的前缀，厂商就可以尝试实现 W3C 涵盖新 CSS 属性的工作草案。在迅速实现新属性的同时，还可以声明它们是过渡的、部分实现的，或者实验性的。总之，后果由使用者自负。

就拿 W3C 推荐的 `transform` 属性为例，标准语法是这样的：

```
transform: skewX(-45deg);
```

然而，由于这个属性还没有完全定案，为保证在大多数浏览器以及它们的实验性实现中能够使用这个属性，应该针对想要支持的浏览器为该属性添加 VSP。每个浏览器只使用各自能理解的属性声明。

```
-moz-transform:skewX(-45deg);    /* Firefox */
-webkit-transform:skewX(-45deg); /* Chrome 及 Safari */
-ms-transform:skewX(-45deg);     /* 微软 Internet Explorer */
-o-transform:skewX(-45deg);      /* Opera */
transform:skewX(-45deg);         /* 最后是 W3C 标准属性 */
```

VSP 的开头是一个连字符，然后是前缀名，接着又是一个连字符，最后是 W3C 属性名。另外要特别注意，在带前缀的属性声明之后还要声明 W3C 标准属性，以备将来有浏览器实现完整的不带前缀的属性时派上用场。这里的 Safari 和 Chrome 都使用相同的 `-webkit-` 前缀，是因为它们都使用 Webkit 渲染引擎。

以下 CSS3 属性必须加 VPS：

```
border-image        translate
linear-gradient     transition
radial-gradient     background*
transform           background-image*
transform-origin
* 针对背景图片或渐变
```

为了节省篇幅，我不会在每个例子中都写全一套带 VPS 的属性声明，而只会提醒大家 VPS 是必要的。随着浏览器的不断更新，有朝一日，VPS 可能就用不着了。有关 CSS3 和 VPS 的最新信息，可以参考这个网站：http://caniuse.com。如果想实现自动添加 VPS，可以使用-prefix-free 腻子脚本（polyfill）——参见本书附录。

3.6.11　背景渐变

渐变就是在一定长度内两种或多种颜色之间自然的过渡。CSS3 之前，必须依赖 Adobe Photoshop 等图形处理软件来制作渐变图，再以背景图片方式添加给元素。而现在，使用 CSS 就可以创造出各种渐变效果了。



渐变是 CSS 帮我们生成的背景图片。添加渐变可以使用 background-image 属性，也可以像后面例子中一样使用简写 background 属性。

渐变分两种，一种线性渐变，一种放射性渐变。线性渐变从元素的一端延伸到另一端，放射性渐变则从元素内一点向四周发散。

下面来看一个简单的线性渐变的例子，HTML 标记如下。

```
<div class='gradient1'></div>
<div class='gradient2'></div>
<div class='gradient3'></div>
```

CSS 规则如下。

```
/*为元素盒子添加样式*/
div {
    height:150px;
    width:200px;
    border:1px solid #ccc;
    float:left;
    margin:16px;
}
/*例 1：默认为从上到下*/
.gradient1 {
    background:linear-gradient(#e86a43, #fff);
}
/*例 2：从左到右*/
```

```
.gradient2 {
    background:linear-gradient(left, #64d1dd, #fff);
}
/*例 3：左上到右下*/
.gradient3 {
    background:linear-gradient(-45deg, #e86a43, #fff);
}
```

图 3-40 展示了三种简单的渐变效果。例 1 声明了一种开始属性和一种结束颜色，这两种颜色会按照默认的方向（从下到下）实现平滑过渡。例 2 起点关键字 `left`，于是渐变方向变成了从左到另一端。例 3 声明了`-45deg`（deg 是“度”），等于把起点从默认的中上设定到了左上。



图 3–40　三种简单的渐变效果

1. 渐变点

渐变点就是渐变方向上的点，可以在这些点上设定颜色和不透明度。通过设定下一个渐变点的颜色值，就可以控制渐变的效果。可以添加任意多个渐变点。渐变点的位置一般使用整个渐变宽度的百分比来表示。图 3-41 展示了使用渐变点后的四种渐变效果。

```
/*例 1：50%处有一个渐变点*/
.gradient1 {
    background:linear-gradient(#64d1dd, #fff 50%, #64d1dd);
}
/*例 2：20%和 80%处有两个渐变点*/
.gradient2 {
    background:linear-gradient(#e86a43 20%, #fff 50%, #e86a43 80%);
}
/*例 3：25%、50%、75%处有三个渐变点*/
.gradient3 {
```

```
    background:linear-gradient(#64d1dd, #fff 25%, #64d1dd 50%, #fff 75%,
#64d1dd);
}
/*例 4：为同一个渐变点设定两种颜色可以得到突变效果*/
.gradient4 {
    background:linear-gradient(#e86a43, #fff 25%, #64d1dd 25%, #64d1dd 75%,
#fff 75%, #e86a43);
}
```

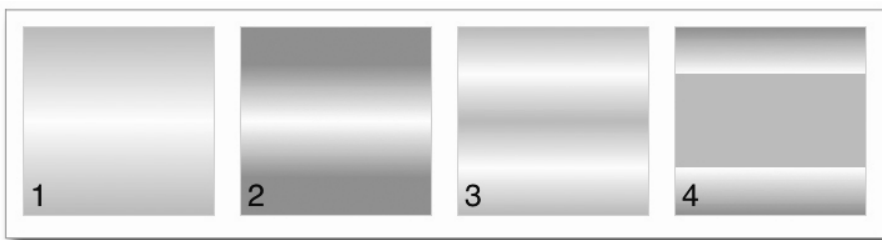


图 3-41　带渐变点的渐变效果

图 3-41 中的例 1 在 50%处包含一个渐变点，因此渐变效果是从开始颜色到渐变点颜色（白色），然后再从渐变点颜色到结束颜色。注意，开始和结束位置如果没有声明，则默认为 0%和 100%。



如果不是使用百分比或其他值声明渐变点的位置，则三种颜色会均匀分布于整个渐变，其实际位置是 0%、50%和 100%。

例 2 演示了起点和终点不是 0%和 100%时的情形。此时，在第一个渐变点（20%）之前，是第一个渐变点声明的实色，而在该点之后，则是从该颜色到下一个渐变点颜色的过渡。同样，在最后一个渐变点（80%）之后，该渐变点的颜色会以实色扩展到元素结束。

例 3 简单展示了相同颜色在几个渐变点之间变来变去的效果。例 4 展示了在同一个渐变点声明两种不同的颜色，能实现一种突变的效果。

2. 放射性渐变

放射性渐变比线性渐变复杂那么一点点，因为可用的控制点多一些。如果你写过程序，从属性值中的括号就可以看出，渐变属性其实是函数。什么是函数？函数可以接收参数，然后根据这些参数来生成渐变。在创建放射性渐变时，可以使用参数指定形状、位置、尺寸、颜色和不透明度。

下面的每一个例子都设定 3 种颜色。

```
.gradient1 {
    background: -webkit-radial-gradient(#fff, #64d1dd, #70aa25);
}
.gradient2 {
    background: -webkit-radial-gradient(circle, #fff, #64d1dd, #e86a43);
}
.gradient3 {
    background: -webkit-radial-gradient(50px 30px, circle, #fff, #64d1dd,
#4947ba);
}
```



这里虽然只声明了-webkit-前缀，但带有其他厂商前缀的属性也是必要的。

在图 3-42 中，例 1 展示了默认的渐变形状，即渐变效果会填充元素，这里的元素是矩形。如果元素是正方形，那渐变就是圆形。

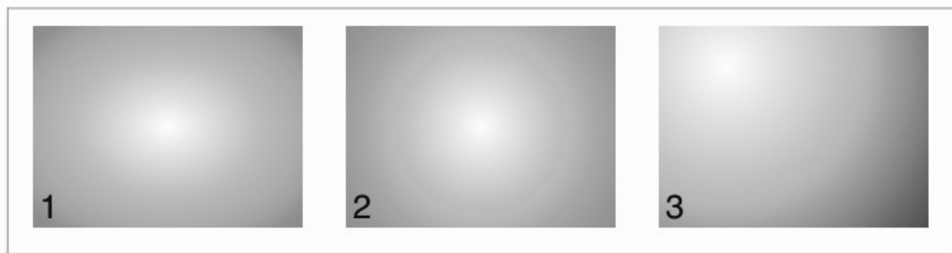


图 3–42　三个三色放射性渐变。第一个是默认的填满图形渐变，第二个是圆形渐变，第三个是指定位置的圆形渐变

例 2 设定了形状关键字 `circle`，于是渐变的形状变得均匀，并在元素最近的边达到了终点，形成了圆形渐变。而长边剩下的区域则填充了终点的颜色。例 3 中的位置参数 `50px 30px` 把渐变的圆心放到了靠近左上角的位置。

好了，通过以上介绍，你应该已经了解了渐变的原理。本书后面几章在讨论 CSS 的实际应用时，还会给出如何使用它们的更多示例。

关于这些例子的详细解释，以及如何控制放射性渐变的位置等内容，建议大家看一看我的电子书 *Visual Stylin' with CSS*，2012 年由 Peachpit Press 出版。

3.7　小结

本章内容不少，我们讲了盒模型的概念，讲了怎么设定外边距、内边距和边框。你也知道了通过浮动可以实现文字绕排图片的效果，可以让块级元素并列排在一行，以及通过清除浮动元素，可以强制包含元素包围浮动的子元素。而定位和显示属性，为我们提供了组织页面元素的手段。本章最后介绍了通过元素的背景层，为元素的内容添加背景颜色、背景图片和渐变效果。

下一章我们得讲一讲 CSS 字体和文本了。当然，也会告诉大家一些创建专业排版效果的技术。

4 字体和文本

网页设计中很多时候都要处理文本，包括段落、标题、列表、菜单和表单中的文本。因此，理解本章要讲的字体和文本属性，对于创造出专业水准的网页排版非常重要。一个网站的品质如何，有时候只要看看它用的字体就能一目了然。如果说图片只是蛋糕上的糖衣，那么排版才是卓越设计的根本。

这一章主要介绍字体和文本，以及与它们相关的 CSS 属性。CSS3 最新的 Web 字体当然不能少，我们会看一看怎么让字体可以随网页下载到用户的浏览器。这样就不必苦巴巴地企盼用户机器上有某种字体了，你完全可以自信地让全世界用户都看到预期的排版效果。

就从字体开始讲起吧。

4.1 字体

网页中的字体有三个来源。

- 用户机器中安装的字体。（直到最近，这些字体还是能在网页中放心使用的唯一一批字体。）
- 保存在第三方网站上的字体。最常见的是 Typekit 和 Google，可以使用 `link` 标签把它们链接到你的页面上。
- 保存在你的 Web 服务器上的字体。这些字体可以使用`@font-face` 规则随网页一起发送给浏览器。

接下来，我们先讨论如何使用那些安装在用户机器中的字体。随后在 4.3 节“Web 字体大揭秘”中再讨论另外两种字体来源。

好吧，以下就是与字体样式相关的 6 个属性。

- font-family
- font-size
- font-style
- font-weight
- font-variant
- font（简写属性）

难道字体和文本不是一回事？

当然不是。请听我解释。

字体是“文字的不同体式”或者“字的形体结构”。对于英文而言，每种字体都是由一组具有独特样式的字母、数字和符号组成的。根据外观，字体可以分为不同的类别（font collection），包括衬线字体（serif）、无衬线字体（sans-serif）和等宽字体（monospace）。每一类字体可以分成不同的字体族（font family），比如 Times 和 Helvetica。而字体族中又可以包含不同的字型（font face），反映了相应字体族基本设计的不同变化，例如 Times Roman、Times Bold、Helvetica Condensed 和 Bodoni italic。

文本就是一组字或字符，比如章标题、段落正文等等，跟使用什么字体无关。

CSS 为字体和文本分别定义了属性。字体属性主要描述一类字体的大小和外观。比如，使用什么字体族（是 Times，还是 Helvitica），多大字号，粗体还是斜体。文本属性描述对文本的处理方式。比如，行高或者字符间距多大，有没有下划线和缩进。

这就是我对字体和文本之间区别的认识。如果你想让文字加粗，或者变斜体，可以设定字体属性。而行高和缩进这种只有对文本块（比如标题和段落）才有意义的样式，则要使用文本属性设定。

4.1.1　字体族

示例：`h2 {font-family:times, serif;}`。

font-family 用于设定元素中的文本使用什么字体。一般来说，应该给整个页面设定一种主字体，然后只对那些需要使用不同字体的元素再应用 font-family。要为整个页面指定字体，可以设定 body 元素的 font-family 属性：

```
body {font-family:verdana, sans-serif;}
```

font-family 是可以继承的属性，因此它的值会遗传给所有后代元素。对于 body 元素来说，这个字体设定可以遗传给标记中所有其他元素。



据测试，font-family 的值（字体名）不区分大小写。但是，你可不能修改 Google 或其他在线字体库生成的字体名，否则很可能无法使用它们提供的定制字体。

既然字体要么来自用户机器，要么来自网上，那么就始终存在某种字体不能在某个网页中使用的可能。正因为如此，设定字体时要给出一组字体来，这组字体叫字体栈。

用字体栈指定本地字体

用户机器上的字体是随操作系统一起安装的，可以由本地应用共享。每种操作系统自带的字体不多，而且用户随时会安装或删除字体，因此我们永远也不敢保证一定能用某种字体来显示网页。为此，在指定文本的字体时，需要多列出几种后备字体，以防第一种字体无效。这个字体的列表也叫字体栈。

简单地说，如果用户机器上有相应字体，那字体栈能保证用户以预期字体来阅读网页文本，如果用户机器上没有相应字体，那么用户还可以使用另一种效果可以接受的字体阅读。

```
body {font-family:"trebuchet ms", tahoma, sans-serif;}
```



如果字体名像 Trebuchet MS 一样多于一个单词（有空格），应该加上引号。

这里的字体栈相当于跟浏览器说："使用 Trebuchet MS 字体显示这个文档，如果机器里没有这款字体，那就使用 Tahoma 代替，如果这款字体也没有，那就随便找一种机器里有的无衬线字体吧。"注意，给 font-family 字体栈的最后一项指定一个通用字体类非常重要，比如 serif、sans-serif，这是一种最保险的方法。

有哪些通用的字体类呢？有以下 5 种：

- serif，也就是衬线字体，在每个字符笔画的末端会有一些装饰线；

- sans-serif，也就是无衬线字体，字符笔画的末端没有装饰线；
- monospace，也就是等宽字体，顾名思义，就是每个字符的宽度相等（也称代码体）；
- cursive，也就是草书体或手写体（在本章后面排版 *The Hound of the Baskervilles*，即《巴斯克维尔庄园的猎犬》的示例中可以看到）；
- fantasy，不能归入其他类别的字体（一般都是奇形怪状的字体）。

使用这些通用字体类的目的，就是确保在最坏的情况下，文档起码可以通过正确的字形来显示。

在选择字体栈的字体时，也要多留心。用过 Dreamweaver 的读者可能知道，每次在 CSS 文件中输入 font-family 时，它都会弹出一个菜单让你选择字体。但每个菜单项中列出的字体相互并不是理想的替代对象。比如，Dreamweaver 会给出如下字体栈的建议：

```
verdana, arial, helvetica, sans-serif;
```

Verdana 的线条比较粗，而且其“x 高度”（x-height）比 Arial 高。如果用户机器上没有 Verdana，就会使用 Arial，这样文本块看起来比预期会小一些。因为每一行能放下的单词多了，文本块高度也可能随之缩短。



“x 高度”指英文字母不包含（字母 d 和 p 都有的）上伸部分和下伸部分的主要区域，恰好 x 没有这些部分，所以就用它的高度来度量。

这里有一个测试技巧，就是把每种字体分别作为字体栈的第一款字体，然后查看页面，就能知道每款后备字体对页面的布局和外观有什么影响了。

Verdana 的后备用 Tahoma 最好，因它们的“x 高度”相同。

```
verdana, tahoma, sans-serif
```

如果想设定一个相对轻爽一点的无衬线字体，可以使用这个字体栈：

```
helvetica, arial, sans-serif
```



关于如何选择字体，大家有空看看这篇文章：http://unitinteractive.com/blog/2008/06/26/better-css-font-stacks/。

下面这个例子展示了一个衬线字体栈，其中第一款字体是用户可能没有的。

```
{font-family:"hoefler text", georgia, times, serif;}
```

在这种情况下，一定要在字体栈后面补上那些大多数操作系统都会内置的字体，比如这里的 georgia、times，最后别忘了通用字体类 serif。

哪些字体是在所有浏览器中都能用的？

这个问题很多人都会问，但却没有固定的答案。不过，以下字体极有可能在 Mac 和 PC 中都安装了。

Serif	Sans-serif	Monospace
Georgia	Arial	Courier New
Palatino/Book Antiqua	Arial Black	Lucida Console/Monaco
Times New Roman	Arial Narrow	
	Tahoma	**Cursive**
	Trebuchet MS	Comic Sans MS
	Verdana	
		Fantasy
		Impact

随着手机和平板电脑的普及，字体可用性越来越难以预测。因此，在字体栈中指定通用字体族也变得越来越重要。如果你想设定一种自己 Web 服务器上的特殊字体，让用户可以随页面下载它，可以参考本章 4.3 节“Web 字体大揭秘”。

4.1.2 字体大小

示例：`h2 {font-size:18px;}`。

浏览器样式表默认为每个 HTML 元素都设定了 `font-size`，因此你在设定 `font-size` 的时候，其实是在修改默认值。由于字体大小在标记层次中是可以继承的，假如你对使用的字体大小单位怎么影响继承不十分了解，那很容易碰到有些字体莫名其妙变大或变小的问题。总的来说，用于设定字体大小的单位有两种，一种是绝对单位，比如像素或点，另一种是相对单位，比如百分比或 em。下面我们就解释这两种单位的区别。

font-size 是可以继承的。换句话说，改变一个元素的字体大小，可能会导致其子元素字体大小成比例地变化。比如说把 body 元素的 font-size 设定为 200%，那么页面中所有元素的文本都会增大一倍。

之所以出现这种效果，是因为浏览器样式表在设定所有元素的字体大小时，使用的都是相对单位 em。比如，h1 被设定为 2em，h2 是 1.5em，p 是 1em。默认情况下，1em 等于 16 像素，这也是 font-size 的基准大小。换算成绝对值，h1 就是 32 像素，h2 是 24 像素，p 是 16 像素。

如果把 body 的字体大小设定为 20px，就是重新设定了基准大小（body 可是所有元素的祖先啊）。因此，h1 会变成 40 像素，h2 会变成 30 像素，而 p 会变成 20 像素。不过，那些以像素之类的绝对单位重新设定字体大小的元素，不会继承祖先元素的字体大小，它们会按照设定的大小显示。

好了，下面看一看设定字体大小的两种方法。

1. 绝对字体大小

使用像素、派卡（pica）或英寸设定字体大小很简单，它们是绝对单位，因此设定多大就多大，与祖先元素的字体大小无关。使用绝对单位的缺点很明显，那就是在需要调整页面所有元素的字体大小时，必须一个一个地修改样式表中的 font-size，相当麻烦。



设定绝对字体大小时，也可以使用关键字值，比如 x-small、medium、x-large，等等。其中，medium 等于基准大小，其他关键字要么小一点，要么大一点。关键字值涵盖的范围太窄，所以使用不广泛，如果你感兴趣，可以看看这篇文章：http://css-discuss.incutio.com/wiki/Using_Keywords（注意大小写）。

简言之，修改 body 元素的字体大小，不会影响页面中以绝对单位控制的元素，但没有设定字体大小的元素则会与 body 的字体大小成比例变化。

2. 相对字体大小

使用百分比、em 或 rem（根元素的字体大小）设定字体大小要复杂一些。如果你给某个元素设定了相对字体大小，则该元素的字体大小要相对于最近的“被设定过字体大小的”祖先元素来确定。

比如以下标记

```
<body>
    <p>This is <strong>very important!</strong></p>
</body>
```

和下面的 CSS

```
p {font-size:.75em;}
strong {font-size:.75em;}
```



如果你想使用 em，但又需要设定具体的像素大小，可以把 body 的 font-size 设定为 62.5%。这样，就等于把基准大小从 16 像素改为 10 像素（16×62.5%=10）。然后，em 与像素的对应关系就十分明确了，比如 1em 等于 10 像素，1.5em 等于 15 像素，2em 等于 20 像素，等等。

p 元素的文本为 12 像素（body 的 16 像素基准大小 × .75 = 12），折合成点单位是 9 点。strong 是 p 的子元素，它的文本是多大呢？相对大小会逐层复合，strong 的大小应该是 16 像素 × .75 × .75 = 9 像素。要熟悉相对单位的这种计算方式，需要多实践。最终你会知道，改变一个元素的相对字体大小，其子元素的字体大小也会同比例变化。



使用相对字体大小，自动调整各层元素。

相对大小的好处也很明显，因为使用相对大小后，通过调整 body 元素的字体大小，可以成比例地改变所有元素的字体大小。或者，至少能通过改变某个祖先元素，只影响它的所有子元素。在反复修改布局设计的时候，这样显然能节省时间。可反过来，毕竟是“牵一发而动全身”的事，所以使用相对字体大小时，必须事先规划好。

使用绝对字体大小是没有办法统一调整的。绝对字体大小只能个别设定，个别修改。当然啦，使用绝对单位的好处呢，就是在祖先元素的字体大小变化时，不会出现意外的“连锁反应”。

不管怎么说，今天设备的屏幕尺寸可谓千差万别，既有特大的显示屏，又有很小的手机屏幕。在这种形势下，更容易缩放的相对大小应该是首选。

3. 关于 rem 单位

CSS3 新增了一个相对单位 rem（root em，根 em），这个单位引起了广泛关注。这个单位与 em 有什么区别呢？区别在于使用 rem 为元素设定字体大小时，仍然是相对大

小，但相对的只是 HTML 根元素。这个单位可谓集相对大小和绝对大小的优点于一身，通过它既可以做到只修改根元素就成比例地调整所有字体大小，又可以避免字体大小逐层复合的连锁反应。目前，除了 IE8 及更早版本外，所有浏览器均已支持 rem。对于不支持它的浏览器，应对方法也很简单，就是多写一个绝对单位的声明。这些浏览器会忽略用 rem 设定的字体大小。下面就是一个例子：

```
/*IE8 及之前版本的 IE 浏览器使用 14 像素*/
p {font-size:14px; font-size:.875rem;}
```



在用户使用“查看 > 文字大小”菜单调整网页文本大小的时候，IE9 及更早版本只能缩放以相对单位设定的文本（使用像素之类的绝对单位设定的文本无法缩放）。换句话说，使用 rem 单位在 IE7 和 IE6 中存在一个小小的副作用，那就是这些浏览器的用户必须使用“查看 > 缩放”来调整整个页面的大小。当然，这也算一个应该升级到现代浏览器的原因吧。

下面我们再讨论其他字体相关的属性。

4.1.3　字体样式

值：italic、oblique、normal。

示例：h2 {font-style:italic;} 。

font-style 设定字体是斜体，还是正体。用 oblique 代替 italic 的结果也一样。

换句话说，font-style 的作用仅仅是通过 italic 把正体设为斜体，或者通过 normal 把斜体设为正体。比如下面的例子：

```
p {font-style:italic;}
span {font-style:normal;}
<p>This is italicized text with <span>a piece of non-italic text</span> in the middle.</p>
```

会得到图 4-1 所示的结果。

This is italicized text with a piece of non-italic text *in the middle.*

图 4–1　font-style 的 normal 值导致一段斜体文本中的几个单词以正体显示

英文中的斜体主要表示强调。假如你真想表示强调，那在 HTML 标记中直接使用<em>标签即可，因为它默认就是斜体。

所谓“常规”值

font-style 有一个 normal 值，中文就是“常规”的意思。 这个值其实不仅 font-style 有，很多其他属性也有，它的作用就是取消所有的特殊样式。什么情况下才会用它呢?

这个值是用来有选择地覆盖某个默认或你设定的全局属性的。比如，h1 到 h6 默认为粗体，如果你想让 h3 以常规字体显示，就需要设定 h3 {font-weight:normal;}。如果你的样式表里声明了 a {font-variant:small-caps;}，那网页中的英文链接都会变成小型大写字母。要是你想让某一组链接仍然以常规的大小写形式显示，可能得另写一个类似这样的声明：a.speciallink {font-variant:normal;}。

4.1.4　字体粗细

可能的值：100、200……900，或者 lighter、normal、bold 和 bolder。

示例：a {font-weight:bold;}。

实际上，前面给出的（CSS 标准规定的）这些数字值没有什么用，对浏览器来说，它只显示 font-weight 属性的两个值：bold 和 normal。由于浏览器对数字值的实现各不相同，所以从常规字体到粗体的切换可能发生在不同的数值上——通常是 400 左右。总之，对于 font-weight 属性来说，最好只用 bold 和 normal 这两个值，效果如图 4-2 所示。

```
p.shows_weight {font-weight:bold;}
p.shows_weight span {font-weight:normal;}
<p class="shows_weight">This is bolded text with <span>a piece of non-bolded
text</span> in the middle.</p>
```

This is bolded text with a piece of non-bolded text **in the middle.**

图 4-2　font-weight 的 normal 值导致一段粗体文本中的几个单词以常规字体显示

粗体的主要作用是表示重要。实际上，HTML 元素 strong 也表示重要，而它的默认样式就是粗体。

4.1.5　字体变化

值：small-caps、normal。

示例：blockquote {font-variant:small-caps;}。

font-variant 属性除了 normal，就只有一个值，即 small-caps。这个值会导致所有小写英文字母变成小型大写字母：

```
h3 {font-variant:small-caps;}
```

以上代码能得到如图 4-3 所示的结果。

THIS TEXT SHOWS THE FONT-VARIANT VALUE SMALL-CAPS.

图 4–3　使用小型大写字母显示的标题，注意标记文本中原来就大写的第一个词的首字母并没有变

我经常将 small-caps 用于::first-line 伪元素，其用法可以在本章后面排版 *The Hound of the Baskervilles*（《巴斯克维尔庄园的猎犬》）的示例中看到。不过，在此提醒读者，对英文来说一定要慎用这种样式，因为大写字母不像小写字母那样有上伸部分和下伸部分作为视觉提示，所以全都使用大写字母会增加辨识难度。

4.1.6　简写字体属性

示例：

```
p {font: bold italic small-caps .9em helvetica, arial, sans-serif;}
<p>Here's a piece of text loaded up with every possible font property.</p>
```

以上使用 font 简写属性的代码能得到图 4-4 所示的效果。

HERE'S A PIECE OF TEXT LOADED UP WITH EVERY POSSIBLE FONT PROPERTY.

图 4–4　一条声明指定字体加粗、斜体、小型大写字母、大小和字体族

没错，font 属性是一个简写形式，通过它只要一条 CSS 声明就可以设定所有字体属性。不过，使用这个简写形式要遵守两条规则，否则浏览器无法正确解释声明的值。

规则一：必须声明 `font-size` 和 `font-family` 的值。

规则二：所有值必须按如下顺序声明。

1. `font-weight`、`font-style`、`font-variant` 不分先后；

2. 然后是 `font-size`；

3. 最后是 `font-family`。



实际上，在设定 `font-size` 属性的同时，可以顺便设定 `line-height`（行高）值。也就是说，字体大小和行高的值可以写在一块，比如 `12px/1.5`。当然，`line-height` 是文本属性，下一节我们会讲到它。

4.2　文本属性

上一节我们讲了字体属性，这一节讲文本属性。如果你想缩进段落，实现 10^6 中的上角标效果，在标题字符间增加一些间距，以及其他一些排版效果，就要用到 CSS 的文本属性。

以下是几个最有用的 CSS 文本属性：

- `text-indent`
- `letter-spacing`
- `word-spacing`
- `text-decoration`
- `text-align`
- `line-height`
- `text-transform`
- `vertical-align`

4.2.1　文本缩进

值：长度值（正、负均可）。

示例：`p {text-indent:3em;}`。

text-indent 属性设定行内盒子相对于包含元素的起点。默认情况下，这个起点就是包含元素的左上角。

给 text-indent 设定正值，文本向右移，得到的是段落首行缩进效果，如图 4-5（中的第一个段落）所示。

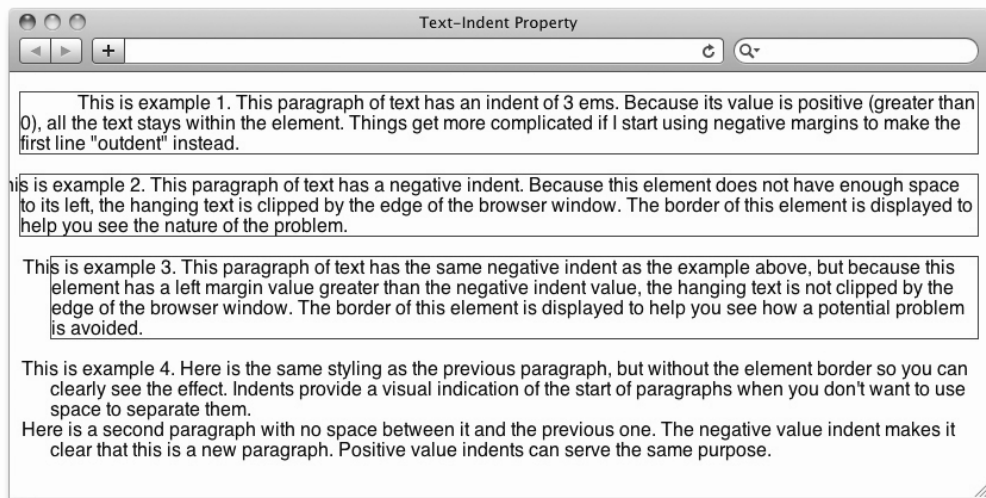


图 4-5　这 4 个段落演示了 text-indent 属性

继承的值与计算的值

这里有一个非常重要的问题，必须请读者注意：text-indent 是可以被子元素继承的。如果你在一个 div 上设定了 text-indent 属性，那么 div 中的所有段落都会继承该缩进值。然而，与所有继承的 CSS 值一样，这个缩进值并不是祖先元素中设定的值，而是计算的值。下面举一个例子说明。

假设有一个 400 像素宽的 div，包含的文本缩进 5%，则缩进的距离是 20 像素（400 的 5%）。在这个 div 中有一个 200 像素宽的段落。作为子元素，它继承父元素的 text-indent 值，所以它包含的文本也缩进。但继承的缩进值是多少呢？不是 5%，而是 20 像素。也就是说，子元素继承的是根据父元素宽度计算得到的缩进值。结果，虽然段落只有父元素一半宽，但其中的文本也会缩进 20 像素。这样可以确保无论段落多宽，它们的缩进距离都一样。当然，在子元素上重新设定 text-indent 属性，可以覆盖继承的值。

文本之“蛇”

在 CSS 中控制文本要理解一个重要概念。CSS 会把元素中的文本放在一个不可见的盒子里，比如对 p 元素中的一段文本，CSS 将其视为很长的一行，只不过在遇到容器边界时会折行。为了看到这个长长的盒子，图 4–6 为包含元素（段落）添加了粗边框，为文本盒子添加了细边框。所有文本属性都会应用给这个细边框的文本盒子。

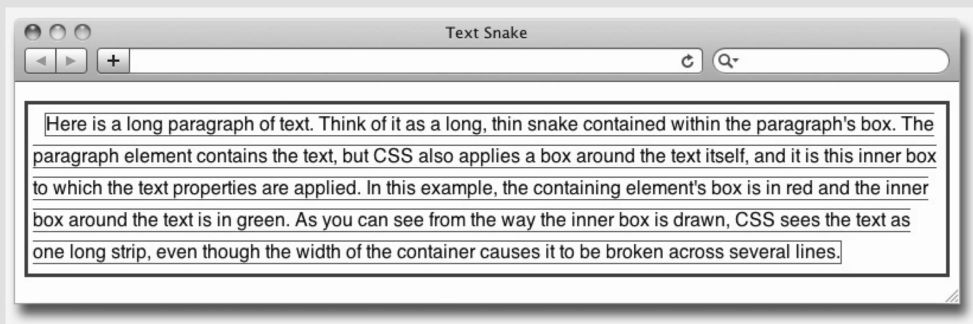


图 4–6　文本被包含在一个细长的盒子中，这个盒子往往因为折行而断开，分布在多行中

在这个例子中，文本元素大致如下：`<p><span>Here is a long paragraph…</span></p>`

相应的样式如下：`p {border:3px solid red;}span {border:1px solid green;}`

这里的文本盒子跨行时是断开的，只有第一行开头和最后一行末尾是封闭的。知道这一点，可以更快地实现你想要的效果。比如，你想缩进段落的第一行，可以使用文本属性 text-indent（如图 4–5 所示），而实际上你缩进的是这个文本盒子的起点位置。后续的行是不会缩进的，因为在 CSS 看来，它们就是一个整体。

如果你想缩进整个段落，可以使用段落的 margin-left 属性。换句话说，你得把整个包含盒子向右移动才行。而你永远要记住，文本属性只应用于这个长长的、细细的、内部的文本盒子，而不是包含元素的盒子。

要是给 text-indent 设定一个负值，那段落的首行会从包含元素的左侧探出头来。如果你想要这种效果，那么一定得给它留出空间。没有空间，而且在左侧还有其他元素的情况下，探出头来的文本就会挡住那个元素。如果左侧是浏览器窗口，那它就会被切掉（如图 4-5 中第二段所示）。解决这个问题的技巧，就是设定一个比负缩进值更长的负左外边距值。图 4-5 中第二段的缩进值为-1.5em，而在图中第三段，我们设定了值为 2em 的左外边距，以下是相应的 CSS 规则：

```
p {
    text-indent:-1.5em;
    margin-left:2em;
    border:1px solid red;
}
```

缩进给人一种专业版式的感觉，也能提示读者每段文本在哪儿开头。请注意，在设定缩进和外边距时要像这里一样使用 em，以便在用户（或你自己）改变字体大小时，它们的长度能够按比例变化。

4.2.2　字符间距

值：任何长度值（正、负值均可）。

示例：`p {letter-spacing:.2em;}`。

`letter-spacing` 为正值时增大字符间距，为负值时缩小间距[①]。无论设定字体大小时使用的是什么单位，设定字符间距一定要用相对单位，以便字间距能随字体大小同比例变化。图 4-7 展示了默认字符间距、增大字符间距和缩小字符间距的例子。用传统排版的术语讲，`letter-spacing` 控制字距（tracking），也就是文本块中所有字符之间的间距。与字距相对的一个排版术语叫字紧排（kerning），紧排只调整两个字符的间距。

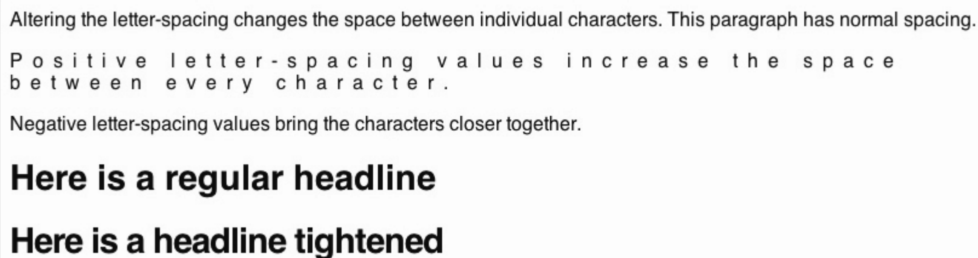


图 4-7　调整字符间距的结果

① `letter-spacing` 对英文字母、汉字及其他字符都起作用，为方便读者理解和记忆，特地译为“字符间距”。另，`letter-spacing` 的值是在浏览器默认值基础上增加或减少的值。——译者注

默认的字符间距在字体变大时会使文本显得松散，因此缩小大标题的字距可以让网页显得更专业。图 4-7 中缩小字距的例子只在默认间距基础上缩小了.05em（1/20），再多的话可能会导致字符交错。

4.2.3　单词间距

值：任何长度值（正、负值均可）。

示例：`p {word-spacing:.2em;}`。

单词间距与字符间距非常相似，区别在于它只调整单词间距，而不影响字符间距。CSS 把任何两边有空白的字符和字符串都视作“单词”①。另外，单词间距比字符间距更容易设定得过大，从而导致文本难以阅读（参见图 4-8）。



只加宽字符间距而不加宽单词间距，不容易看出有什么差别。此时，再加上一点单词间距效果会很好。

Altering the word-spacing changes the space between individual words. This paragraph has normal spacing.

Negativewordspacingvaluesbringthewordsclosertogetherandcanreducereadability.

Positive word spacing values increase the space between every word.

Here is a regular headline

Thisheadlinehasnegativewordspacing

This headline has positive word spacing

图 4–8　单词间距分别为默认值、负值和正值时的几个段落和标题

4.2.4　文本装饰

值：`underline`、`overline`、`line-through`、`blink`、`none`。

示例：`.retailprice {text-decoration:line-through;}`。

① 纯汉字文本一段之中没有空格，因此 `word-spacing` 对中文网页几乎没有用，但对中英混排段落可能有用。——译者注

除了 blink 之外，text-decoration 属性其他值的效果如图 4-9 所示。blink 是为文本添加闪烁效果的，实际上很讨厌，应该少用，最好不用。

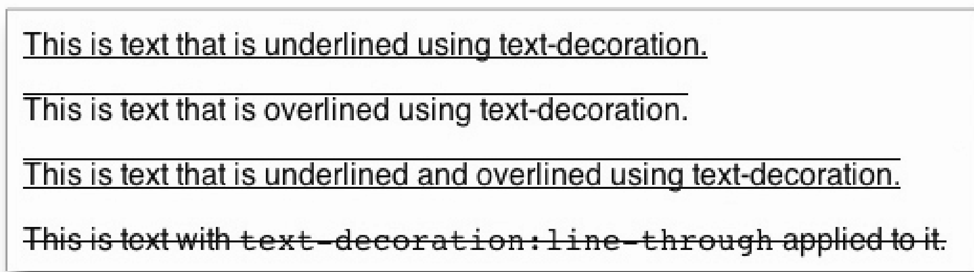


图 4-9　这里展示了 text-decoration 属性各个值的效果，其中使用最多的是用它控制链接的下划线



必须注意，上网的人都习惯了把带下划线的文本当成链接。如果你给本来不是链接的文本加上下划线，很容易导致困惑和无效点击。

文本装饰最主要的应用是控制链接的下划线。下面这个例子展示了怎么通过 text-decoration 去掉导航条中链接的下划线。导航条中的文本显然是可以点击的，下划线徒增混乱而已。不过，在鼠标悬停状态下加上下划线，则是一种有效的视觉反馈。

```
nav a {text-decoration:none;}
a:hover {text-decoration:underline;}
```

4.2.5　文本对齐

值：left、right、center、justify。

示例：p {text-align:right;}。

text-align 属性只有 4 个值，left、right、center 和 justify，控制着文本在水平方向对齐的方式。其中，center 值也可以用来在较大的元素中居中较小的固定宽度的元素或图片。图 4-10 展示了 text-align 属性 4 个值的实际效果。

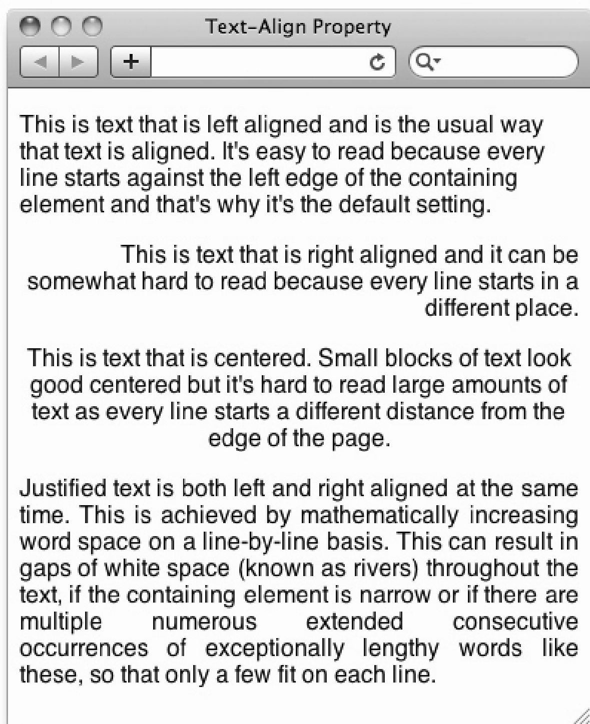


图 4-10　`text-align` 的 4 个值：左对齐、右对齐、居中对齐和两端对齐

4.2.6　行高

值：任何数字值（不用指定单位）。

示例：`p {line-height:1.5;}`。

CSS 通过 `line-height` 属性实现了印刷行业中常说的加铅条（leading）[①]。铅条在活字排版时代用于在文本行之间增加间距。



相信不少读者是头一次听说“铅条”。铅活字印刷时代，两行字模之间的空白间距就是通过向其中加入细铅条实现的。

CSS 中的行高平均分布在一行文本的上方和下方。举个例子，如果字体大小是 12 像

① 关于传统印刷行业中“铅条”这个术语的更多有意思的背景介绍，请参考图灵社区文章“你未必知道的 CSS 故事：揭开 leading 的面纱”：http://www.ituring.com.cn/article/18076。——译者注

素，行高是 20 像素，则浏览器会在文本上方和下方各加 4 像素的空白，以凑足 20 像素的行高。[①]

对于类似标题的一行文本来说，line-height 增加的空白就像外边距一样，较大的标题（如 h1 和 h2）往往有较大的默认行高。这一点应该记住，因为有时候即使把外边距和内边距全去掉，标题上下仍然会有空白。如果想把这些空白也去掉，那就只能把文本的行高设定为比字体高度（就是字体大小）还要小，比如设定为小于 1 的值。

图 4-11　在标准行高的基础上有所变化，是让页面与众不同的一种简单方式

如图 4-11 所示，修改默认行高最简单的方式就是使用 font 快捷属性，以复合值的形式把 font-size 和 line-height 值写在一块，比如：

```
div#intro {font:1.2em/1.4 helvetica, arial, sans-serif;}
```

就这个例子来说，行高就是字体大小 1.2em 的 1.4 倍。注意这里不用给 line-height 值指定单位，只要一个数值就可以。此时，浏览器会用计算得到的 1.2em 的像素数乘以 1.4，结果就是行高。如果后来你把字体大小增大到 1.5em，则行高仍然是 1.5em 计算值的 1.4 倍。在设定行高时如果使用了绝对单位（如像素），那将来增大字体有可能导致行与行之间重叠。

4.2.7　文本转换

值：none、uppercase、lowercase、capitalize。

示例：p {text-transform:capitalize;}。

① 注意，与外边距不同，行高是不叠加的。——译者注

text-transform 属性用于转换元素中文本的大小写，它可以设定英文文本首字母大写、全部字母大写和全部字母小写。图 4-12 展示了 text-transform 属性几个值的作用。

This is regular text that has not been transformed.

This Is Text That Is Capitalized Using The Tranform Capitalize Value. There Are No Capital (Uppercase) Letters In The Markup.

THIS IS TEXT STYLED WITH THE TRANSFORM UPPERCASE VALUE. THERE ARE NO UPPERCASE LETTERS IN THE MARKUP.

this is text that is written in uppercase but the lowercase value transforms it into lowercase text.

图 4–12　使用 text–transform 可以实现类似英文报纸标题的效果

capitalize 值会将每个词的首字母转换为大写，这种效果在很多广告、报道、杂志的标题中很常见。但人为的情况下，像“of”、“as”，还有“Tom and Jerry Go to Vegas.”中的“and”这些词首字母是不大写的，而在 CSS 中，则一律大写。因此用 CSS 设定 capitalize 值，你看到将是“Tom And Jerry Go To Vegas.”。



要想实现小型大写字母的首字母放大效果，可以再加上 font-variant:small-caps 声明。

4.2.8　垂直对齐

值：任意长度值以及 sub、super、top、middle、bottom 等。

示例：span {vertical-align:60%;}。

vertical-align 以基线为参照上下移动文本，但这个属性只影响行内元素。如果你想在垂直方向上对齐块级元素，必须把其 display 属性设定为 inline。vertical-align 属性最常用于公式或化学分子式中的上标和下标，比如 x^4-y^{-5} 或 N_3O，或者用于文本中脚注的角标，比如把星号变成上角标。我个人不太喜欢浏览器为上标设定的默认样式，我觉得有点大，位置也有点高（下标的位置有点低）。写一点样式就可以得到更好的效果，而且能够跨浏览器保持相同比例。

就以下面的 HTML 为例：

```
<h4>Default <code>sub</code> and <code>sup</code> styles</h4>
<p>Enjoy mountain spring H<sub>2</sub>0. It’s 10<sup>5</sup> times better than
tap<sup>&dagger;</sup> water!</p>
```

```
<p class="customsmall"><sup>&dagger;</sup><em>This means water provided
through a municipal distribution system</em></p>
<h4>Custom <code>sub</code> and <code>sup</code> styles</h4>
<p class="custom">Enjoy mountain spring H<sub>2</sub>O. It’s 10<sup>5</sup>
times better than tap<sup>&dagger;</sup> water!</p>
<p class="customsmall"><sup>&dagger;</sup><em>This means water provided
through a municipal distribution system</em></p>
```

配上下面的 CSS 规则：

```
p.custom sub {font-size:60%; vertical-align:-.4em;}
p.custom sup {font-size:65%; vertical-align:.65em;}
p.customsmall {font-size:.8em; vertical-align:1em;}
```

就可以得到如图 4-13 下面所示的效果。

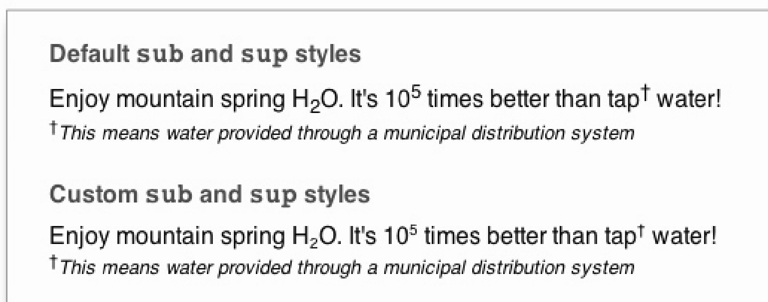


图 4-13　上标和下标的垂直位置和字体大小都不一样

虽然 HTML 标签`<sup>`和`<sub>`有默认的上标和下标样式，但重新设定一下`vertical-align`和`font-size`属性能得到更美观的效果。

目前我们已经介绍完了 CSS 字体和文本属性，接下来该讲一讲让网页下载字体了。

4.3　Web 字体大揭秘

目前，使用`@font-face`规则在网页中嵌入可下载字体的 CSS 功能，已经得到了浏览器广泛支持。

`@font-face` 规则为设计师提供了系统自带字体以外的广泛选择。换句话说，浏览器可以从 Web 服务器下载字体，就意味着不必再依赖用户机器中的字体，而且也可以

确保用户能够看到 CSS 中设定的字体。

设定 Web 字体的方式有以下三种。

- 使用 Google Web Fonts 或 Adobe 的 Typekit 等公共字体库。
- 使用事先打好包的`@font-face`包。
- 使用 Font Squirrel 用你自己的字体生成`@font-face`包。

下面，我们从最简单的方法——使用字体库，开始介绍。

4.3.1 公共字体库

Google Web Fonts 和 Adobe 的 Typekit 是目前最大的两个在线公共字体库，前者托管着大约 500 多类字体，后者包含 739 个字体族，以订购方式提供访问。这两个字体库的用户界面都十分友好。

来看看 Google Web Fonts 的使用步骤。打开 http://www.google.com/webfonts，找到想要的字体，单击 Add to Collection 按钮，然后单击页面底部的 Use 按钮（如图 4-14 所示）。然后，Google 就会生成一个指向你刚刚选定字体的`<link>`标签，直接把它复制粘贴到你的 HTML 文件中即可。

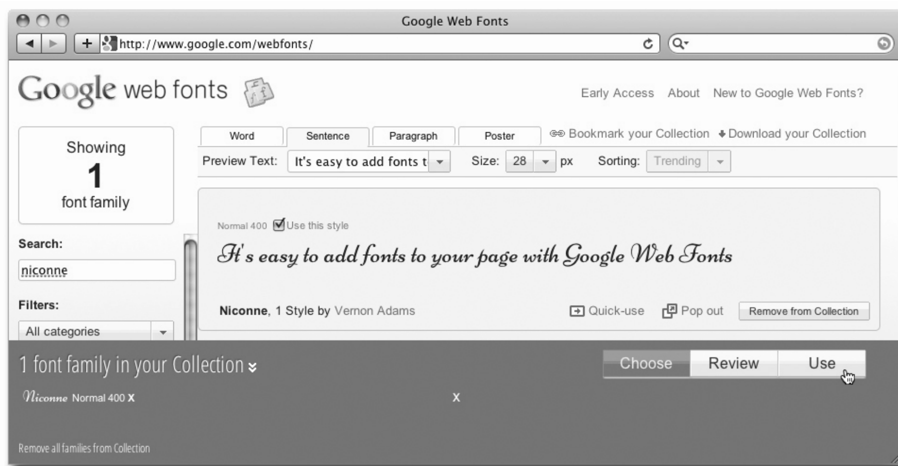


图 4-14　Google 已经把 Niconne 字体放到我的集合里，然后它会给我生成一个指向该字体的链接

一行代码可以链接多款字体。比如下面这个`<link>`标签就引用了 Anton、Niconne 和 Prata 字体。

```
<link href='http://fonts.googleapis.com/css?family=Anton|Niconne|Prata'
rel='stylesheet'  type='text/css'>
```

把这个链接添加到页面的<head>标签中之后，就可以像使用其他字体一样使用这些 Web 字体了。用户打开网页时，浏览器就会从 Google 的服务器得到相应字体。比如

```
h3 {font: 20px "Prata", serif;}
```

的效果如图 4-15 所示。

This text is set in the Prata font.

图 4-15　用户可以看到 Prata 字体的标题了

在线字体库能够迅速而有效地扩展有限的系统字体。在页面中使用 Google 字体就是几分钟的事儿，而且关键是这样一来，你的用户就真的能看到你设定的字体了。

4.3.2　打包的@font-face包

在网页中嵌入字体的第二种方式是使用@font-face 规则，前提是可以从你的网站或第三方 Web 服务器下载到相应的字体。以这种方式提供的字体，会在使用该字体的页面第一次加载时被浏览器下载并缓存起来，以后就不用下载了。但是，除了显示网页之外，用户不能将这种字体用于其他用途。

使用@font-face 规则相对来说要麻烦一点，不过却能做到想用什么字体就用什么字体。字体一般都有许可限制，你要么花钱买字体，要么选择不需要付钱且允许嵌入网页的字体。

使用@font-face 的一个问题是不同浏览器要求的字体格式不一样。比如，Firefox、Webkit 核心的浏览器（Safari 和 Chrome），以及 iOS 4.1 版之后的移动 Safari 使用 OTF（OpenType）或 TTF（TrueType）字体。Internet Explorer 使用 EOT（Embedded OpenType）。另外，iOS 4.1 之前版本中的移动 Safari 以及其他浏览器使用 SVG（Scalable Vector Graphics）格式。虽然字体格式不同，但它们往往以现成的字体包的形式存在，或者可以使用本地字体来生成字体包（提醒你一下，自己生成字体包首先要获得许可）。

Font Squirrel（http://www.fontsquirrel.com）提供了很多现成的字体包，每个字体包中

都包含所有必要格式的字体和为每款浏览器提供正确格式的 CSS 代码。Font Squirrel 还有一个转换程序，能够把你上传的字体转换成字体包。

下面就是 Font Squirrel 为 Ubuntu Titling Bold 字体生成的@font-face 代码。对于其他来源的字体，这种格式也是适用的。

```
@font-face {
    /*这就是将来在字体栈中引用的字体族的名字*/
    font-family:'UbuntuTitlingBold';
    src: url('UbuntuTitling-Bold-webfont.eot');
        src: url('UbuntuTitling-Bold-webfont.eot?#iefix')
        format('embedded-opentype'),
        url('UbuntuTitling-Bold-webfont.woff')
        format('woff'),
        url('UbuntuTitling-Bold-webfont.ttf')
        format('truetype'),
        url('UbuntuTitling-Bold-webfont.
        svg#UbuntuTitlingBold') format('svg');
    font-weight: normal;
    font-style: normal;
}
```

把以上代码添加到网页中之后，就可以使用 font-family 以常规方式引用该字体了。引用字体时要使用@font-face 规则中 font-family 属性的值作为字体族的名字。



Web 专家 Paul Irish 写过一个跨浏览器@font-face 的"笑脸版"，可以保证用户机器中万一安装了同名字体时也不会混淆，详细内容请参考这篇文章：http://paulirish.com/2009/bulletproof-font-face-implementation-syntax/。

4.3.3　生成@font-face包

有时候，你可能希望在自己的网页中使用一种特殊字体，比如客户自己公司指定了一种字体，必须用在网站和设计中。今天，只要你获得了把该字体转换为 Web 字体使用的授权（请查看许可协议或联系字体设计商），就可以使用 Font Squirrel 的转换程序（http://www.fontsquirrel.com/fontface/generator）把它转换成@font-face 字体包。只要简单几步，几分钟时间，就可以得到一个能放到你自己 Web 服务器上供下载的@font-face 字体包。



如果想深入理解@font-face 规则，建议读一读 Tim Brown 的博客文章“How to use @font-face”：http://nicewebtype.com/notes/2009/10/30/how-to-use-css-font-face/。

在讨论设计文字版式的例子之前，还有两件事要跟大家交待。除非所有浏览器都统一支持一种字体格式（很可能是 OpenType），否则我们必须得面对多种字体格式并存的问题。关于多种字体格式的@font-face 规则如何写，以及如何保证 Internet Explorer 取得必要的.eot 格式的字体，可以参考 Fontspring 博客的文章：http://www.fontspring.com/blog/fixing-ie9-font-face-problems，Fontspring 也销售字体，允许以@font-face 方式使用。

从 Web 诞生的那一天起，设计师能够在 PC 和 Mac 上使用的字体就是有限的（使用独特的字体需要太多的额外努力）。经过长久的等待，所有浏览器——包括 IE9 及更高版本，终于都支持@font-face 了。Web 设计师从此也能像平面设计师一样随心所欲地使用字体了。对于那些不支持@font-face 的旧版本浏览器，解决办法也简单，就是在字体栈中同时列出那些用户机器中最可能有的字体，最好是紧跟在首选的嵌入字体之后。

4.4　文字版式

现在该检验一下我们实际应用字体和文本样式的能力了。在本章最后这一节，我们要动手循序渐进地做三个网页排版的例子。

文字排版讲求匀称，一般是由看不见的网格，框定页面文字的走向和布局。匀称的版式可以增强页面的可读性。

下面我们先来看一个基本的、简单的例子。在这个例子中，我们先不用网格，只是基于元素中的字体大小按比例地设定元素间距。这个练习可以告诉你怎么高效地实现想要的效果。

4.4.1　简单的文本布局

我们在第 1 章就已经介绍过了，浏览器为标题、段落、列表及其他元素默认应用的字体大小并不一致，而且垂直外边距也太大。为了示范怎么覆盖默认样式，获得匀称的版面，需要以下包含常用文本元素的 HTML 标记。

```
<article>
    <h1>CSS</h1>
    <p>CSS stands for Cascading Style Sheets. CSS controls the presentational
        aspects of your Web pages.</p>
    <h2>Block-Level Elements</h2>
    <p>Block-level elements stack down the page. They include:</p>
    <ul>
        <li><code>header</code></li>
        <li><code>section</code></li>
        <li><code>h1, h2, etc.</code></li>
    </ul>
    <h2>Inline Elements</h2>
    <p>Inline elements sit next to each other, if there is room. They include:</p>
    <ul>
        <li><code>img</code></li>
        <li><code>a</code></li>
        <li><code>em</code></li>
    </ul>
    <blockquote>
        <q>Typography maketh the Web site.</q><cite>CWS</cite>
    </blockquote>
</article>
```

图 4-16 展示了以上标记在浏览器中的效果。

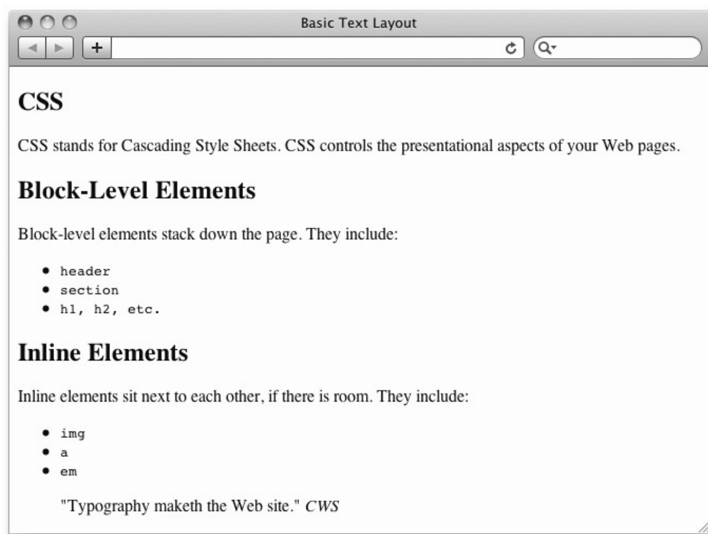


图 4–16　没有添加样式的标记没有什么吸引力

下面我们来写一些样式，让版面看起来更舒服。首先，去掉元素的外边距，设定主字体大小，为包含所有文本元素的 article 应用样式，从视觉上突出它作为容器的角色，然后将它在页面上居中。

```
/*删除所有元素的外边距*/
* {margin:0; padding:0;}
/*设定主字体族和字体大小*/
body {font:1.0em helvetica, arial, sans-serif;}
/*居中显示盒子*/
article {width:500px; margin:20px auto; padding:20px; border:2px solid #999;}
```



值为 1.0em 的 font-size 只是显式地声明了默认字体大小，并没有修改什么。但为了将来修改页面中所有文本的字体更方便，必须在 font 简写属性中同时设定字体大小，而且要使用相对单位 em。

图 4-17 展示了此时在浏览器中看到的结果。

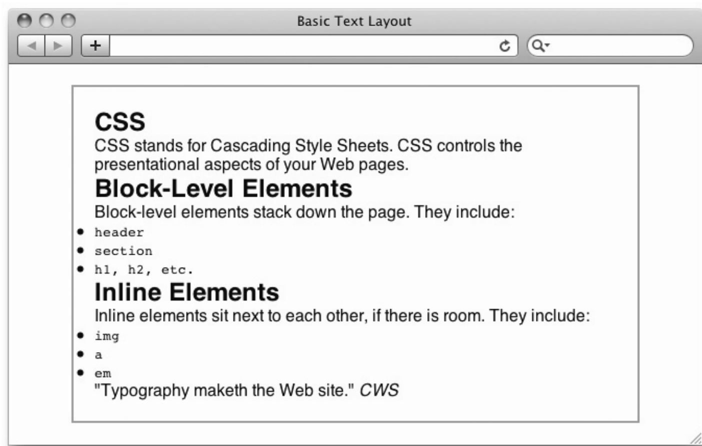


图 4-17　删除默认的外边距显著减少了内容的高度

接下来，需要巧妙地安排一下元素间的垂直距离。另外，去掉默认外边距后列表项目的符号也跑到了外面，这里一块儿修复。

```
/*标题周围的空白*/
h1, h2, h3, h4, h5, h6 {line-height:1.15em; margin-bottom:.1em;}
/*文本元素周围的空白*/
p, ul, blockquote {line-height:1.15em; margin-bottom:.75em;}
/*缩进列表*/
ul {margin-left:32px;}
```

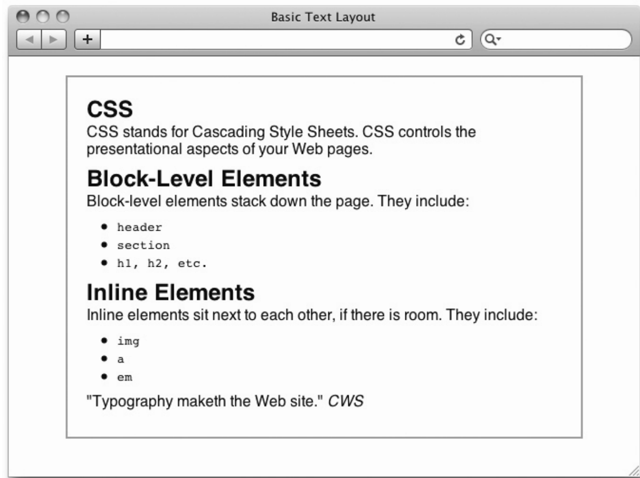


图 4-18 为段落下方添加了空白

如图 4-18 所示，我们把所有元素的行高都缩小了，让它们只比文本稍微高一点点。为什么呢？因为 line-height 会平均分布在文本上下，而我只想通过下外边距在每个元素下方添加空白。但同时，还要留出少量行高，以防段落（和跨行的标题）中相邻的行紧挨上。

为此，这里只设定了两处外边距，而实际的空白要相对于元素的字体大小来计算。标题的下外边距非常小（等于各自字体大小的 10%），因此后面的文本会离它很近。文本元素的下外边距大一些（等于各自字体大小的 75%），以便布局中有较明显的空白。

最后，我想让各级标题也更均衡一些，保证较大的标题突出，最小的标题也不至于不明显，同时也增大了 code 元素的字体。

```
/*调整标题文本*/
h1 {font-size:1.9em;}
h2 {font-size:1.6em;}
h3 {font-size:1.4em;}
h4 {font-size:1.2em;}
h5 {font-size:1em;}
h6 {font-size:.9em;}
/*调整段落文本*/
p {font-size:.9em;}
/*调整代码文本（默认值太小了）*/
code {font-size:1.3em;}
```

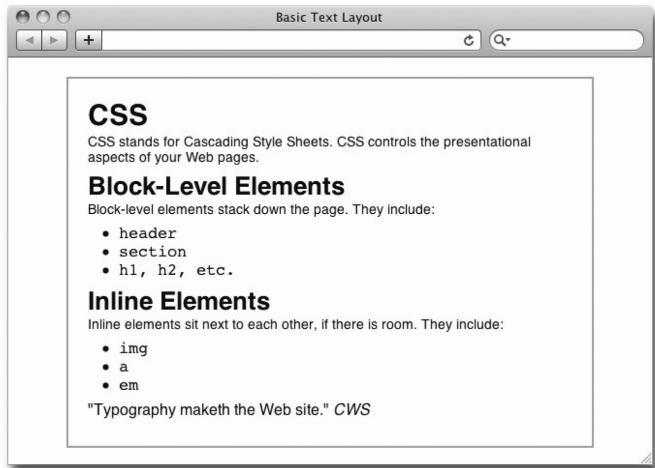


图 4-19　标题和代码都加大了，而且页面也更有层次感

这个例子虽然又简单又基础，但它却展示了如何通过最少量的文本样式来提高页面版式的专业水准和内容的可读性，最终结果如图 4-19 所示。下面我们再看看怎么借助网格来创造出更有吸引力的页面布局。

4.4.2　基于网格排版

借助网格可以保证布局匀称，同时让页面看起来也很流畅。因为本章主要介绍网页版式，所以我们只讨论借助网格创建垂直的文本流。

在下面的例子中，我会基于垂直的 18 像素网格来规划布局，并让页面中的每一个元素都按网格线对齐。还记得可以在元素的背景上添加图片吧，我们可以临时在 body 元素的背景上添加一张图片，让它为整个页面生成辅助线。

我使用 Adobe Fireworks（你可以选择自己最称手的软件）创建了一个 100 × 18 像素的矩形，并在它的底部画了一条灰线。然后将其保存为.png 格式（.jpg 或.gif 格式也没有问题），并命名为 grid_18px.png.，图 4-20 展示了这张图片（为清楚起见，我把它放在了一个深色背景上）。



图 4-20　用作页面背景的小拼贴图，长方形底部有一条灰线

把这张图片放到 body 元素的背景上：

```
/*添加网格线*/
body {background-image:url(images/grid_18px.png);}
```

然后它就会像贴瓷砖一样“贴”满了整个页面，如图 4-21 所示。

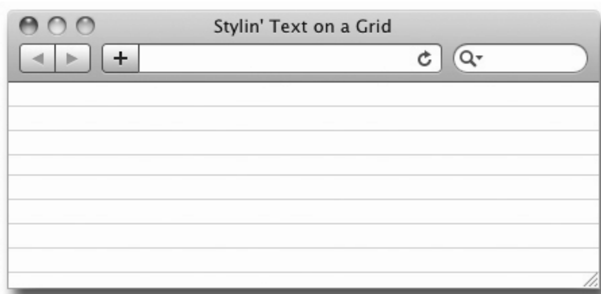


图 4-21 为 body 元素应用横格背景，以便垂直排版文字

生成水平的网格背景后，就可以依照网格来定位文本元素了。

这个例子只使用了一些常见的文本元素，但只要理解了背后的原理，写出一个让 HTML 文本元素“对齐网格”的样式表一点也不难。而有了这样的样式表，整个网站的布局就有了基础。

这个例子包含如下所示的简单段落：

```
<p>In traditional typography, text is composed…</p>
```

CSS 规则如下：

```
/*去掉所有元素的内边距和外边距*/
* {margin:0; padding:0;}
body {
    /*添加网格背景*/
    background-image:url(images/grid_18px.png);
    /*设定字体*/
    font:100% helvetica, arial, sans-serif;
    /*加宽左右外边距，得到一栏的雏形*/
    margin:0 40px 0;
}
p {
    /*设定字体大小*/
```

```
    font-size:13px;
    /*将行高设定为等于网格高*/
    line-height:18px;
}
```

注意，这里把文本的 `line-height` 设定为网格间的距离——18 像素。在去掉默认外边距和内边距的情况下，这样就可以确保每一行都相距 18 像素，如图 4-22 所示。

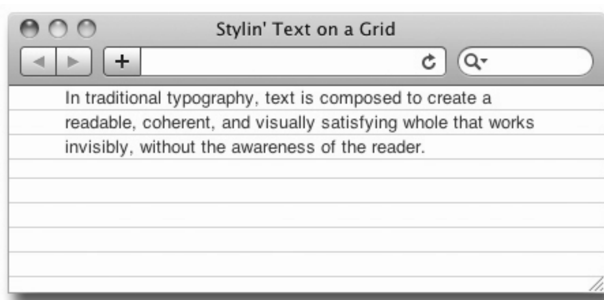


图 4-22　18 像素的行高让文本行的间距与网格间距恰好一致

再给容器 body 元素添加 4 像素的内边距，以便把元素向下推到文本基线与网格对齐的位置。只要让第一个元素与网格对齐，其他元素就都能对齐了。实际上，我给 body 上方添加了 22 像素（4+18）的内边距，以便上方能有一定的空间：

```
padding-top:22px;
```

之后，给段落添加一条声明：

```
p {
    /*设定字体大小*/
    font-size:13px;
    /*把行高设定为等于网格高度*/
    line-height:18px;
    margin-bottom:18px;
}
```

这条声明恰好能在每个段落之间加上一个空行。再加上一段文本，就能更清楚地看这两处修改的效果了（见图 4-23）。

现在，文本与网格对齐了，段落之间的间距也合适了。接着要设定其他文本元素的字体大小。首先把 h3 设定为 18 像素，而为了让它恰好在网站中占一行，必须得把它的 `line-height` 也设定为 18 像素。为了测试，我们在两段之间加入一个标签。

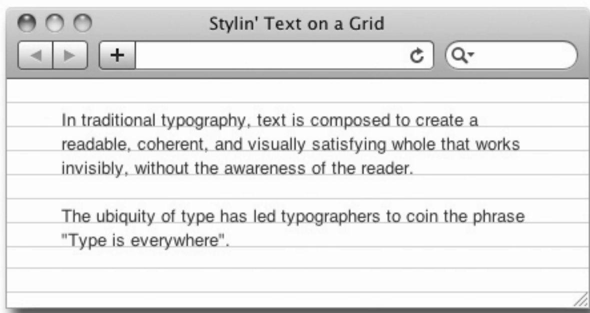


图 4-23　为 body 添加了内边距之后，文本与网格完美地对齐了

```
<p>In traditional typography, text is composed…</p>
<h3>Type for Every Use</h3>
<p>The ubiquity of type has led typographers…</p>
```

针对这个标题的 CSS 规则如下：

```
h3 {font-size:18px; line-height:18px;}
```

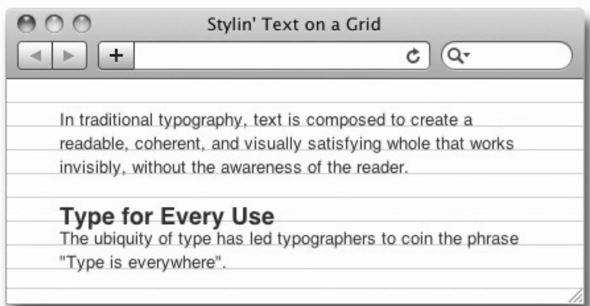


图 4-24　h3 元素的基线比网格线稍微低一点

如图 4-24 所示，标题的基线比网格低几个像素。然而奇怪的是，下方的段落并没有因此向下挪动相应距离。这是为什么呢？因为标题的 `line-height` 是正确的，只是由于它现在的字体大小以及字体的因素，导致文本在内部稍微有一点偏移。纠正这个问题的 CSS 如下：

```
h3 {
    font-size:18px;
    line-height:18px;
    margin-top:-2px;
    margin-bottom:2px;
}
```

上方的负外边距把文本向上拉，下方同样数量的正外边距用来抵消它，保证后面元素的位置不受影响（参见图 4-25）。

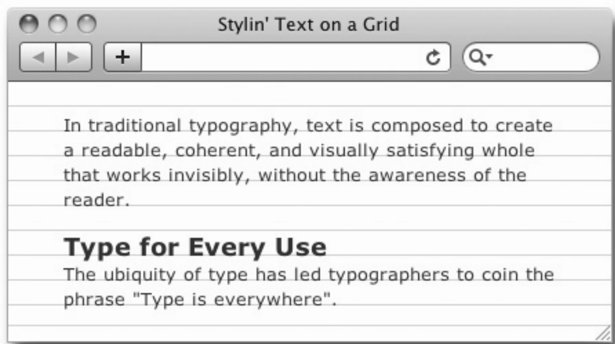


图 4-25　少许上负外边距和同等数量的下正外边距把标题提升到了恰好与网格对齐的位置

我们还需要一种类似技巧来对齐比网格高的元素（通常是标题）。为此，要添加一个 24 像素的 h1 元素。显然，24 像素的文本占据的空间比一行网格要高，因此给它设定的 line-height 是 36 像素，也就是两行网格的高度。然后，就是把 h1 元素放到它应该出现的地方——页面的开头。

```
<h1>Typography</h1>
<p>In traditional typography…</p>
```

当前给它设定的 CSS 如下：

```
h1 {font-size:24px; line-height:36px;}
```

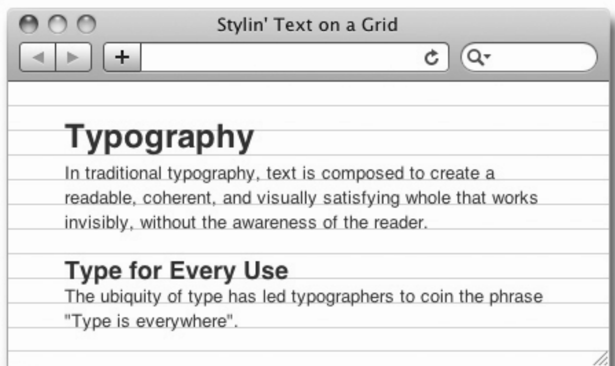


图 4-26　由于 line-height 等于两倍网格行高，所以标题没有在网格线上

这个占两行网格的大标题看起来不太舒服。如果把它移动到下方最近的线上，那么其字母下伸部分就会接触到段落文本，所以我要把它向上移动。经过一番试错，我决定把它向上移动 13 像素。



另一种思路是把标题文本的基线放在两行网格线之间一半的位置。那会实现一种不同的风格。不过，无论如何都要保证后面的元素与网格对齐。

```
h1 {
    font-size:24px;
    line-height:36px;
    margin-top:-13px;
    margin-bottom:13px;
}
```

这样，h1 下方与文本之间就空出了一定距离（参见图 4-27）。对于级别低一点的标题也可以应用同样的技巧，但我个人还是倾向于认为标题与其下方元素再接近一些才舒服。

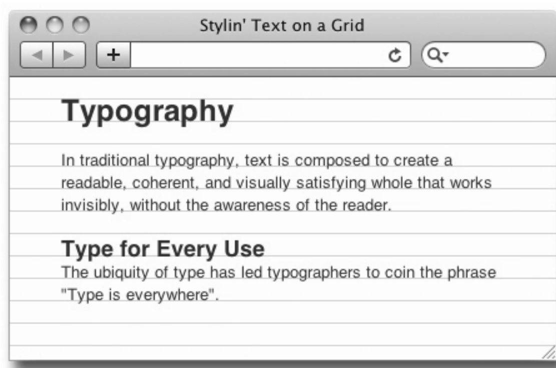


图 4-27　h1 元素的基线与网格线对齐了

在这个练习最后，我们再加几个不同字体大小的标题、一个无序列表和一个 blockquote 元素。看一看把网格去掉之后页面是什么样子。



这个练习的 HTML 代码可以在本书网站下载：http://www.stylinwithcss.com。

```
* {margin:0; padding:0;}
body {font:100% helvetica, arial, sans-serif; backgroundimage:
url(images/grid_18px.png); margin:0 20px 0;
padding:21px;}
```

```
p {font-size:14px; line-height:18px; margin-bottom:18px;}
h1 {font-size:24px; line-height:36px; margin-top:-13px;
    margin-bottom:13px;}
h2 {font-size:18px; line-height:18px; margin-top:-2px;
    margin-bottom:2px;}
h3 {font-size:16px; line-height:18px; margin-top:-2px;
    margin-bottom:2px;}
ul {margin-bottom:18px;}
li {font-size:13px; list-style-type:none; padding:0 20px;
    line-height:18px;}
a {color:#777; text-decoration:none;}
blockquote {font-size:12px; line-height:18px; paddingtop:
            2px; margin-bottom:16px;}
```

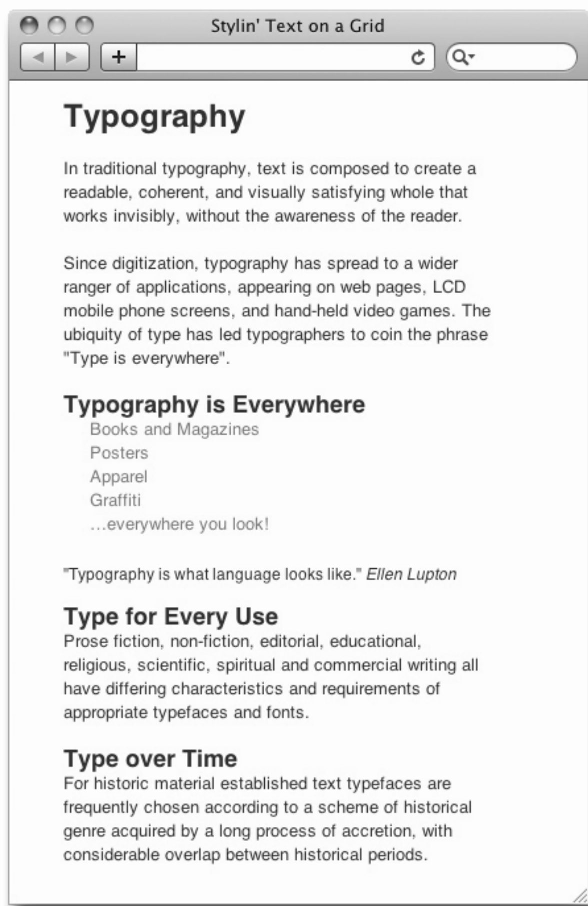


图 4-28　基于 18 像素网格的页面布局

如图 4-28 所示，利用网格辅助排版之后的布局还是不错的。从技术角度讲，如果你基于网格为一个网站设计了版式，而后将其交给了其他人维护，那这个网站的版式就能始终保持一致，无论页面中包含什么元素，以及以什么顺序包含这些元素。

4.4.3 经典的排版练习

本章最后，我们节选 *The Hound of the Baskervilles*（《巴斯克维尔庄园的猎犬 》）[①]中的部分文本（为了示例需要进行了删减）来做一个排版练习。这个练习将会用到本章讲过的很多字体和文本属性。完成这个练习后，你就能掌握实现专业排版的很多技术，包括使用 HTML 实体、调整字母和单词间距、首字下沉、垂直网格（这一次是 24 像素）以及可下载字体。

示例的 HTML 标记非常简单，只有两个标题、几个段落和一个块引用。

```
<h2>an excerpt from</h2>
<h1>The Hound of the Baskervilles</h1>
    <p>Holmes stretched out his hand for the manuscript and flattened it upon
       his knee. &ldquo;You will observe, Watson, the alternative use of the
       long s and the short. It is one of several indications which enabled me
       to fix the date.&rdquo; At the head was written: &ldquo;;Baskerville
       Hall,&rdquo; and below in large, scrawling figures: &ldquo;;1742.&rdquo;</p>
    <p>&ldquo;;It appears to be a statement of some sort.&rdquo;</p>
    <p>&ldquo;;Yes&mdash;it is a statement of a certain legend which runs in the
       Baskerville family.&rdquo;</p>
    <blockquote>
    <q>Of the origin of the Hound of the Baskervilles there have been many
       statements, yet as I come in a direct line from Hugo Baskerville, and
       as I had the story from my father&hellip;</q>
    </blockquote>
    <p>When Dr. Mortimer had finished reading this singular narrative he pushed
       his spectacles up on his forehead and stared across at Mr. Sherlock
       Holmes.</p>
```

这段标记包含了几个 HTML 实体，它们代表的是几种标点符号。其中，有表示对话开始的左双引号（“）、表示对话结束的右双引号（”）、表示省略的省略号（…）和表示暂停或代替括号的破折号（—）。

①《巴斯克维尔庄园的猎犬》是阿瑟·柯南·道尔最得意的长篇杰作之一，堪称福尔摩斯探案故事的代表作。
——译者注

HTML 实体引用

以下网站中列出了 HTML 实体：

http://htmlhelp.com/reference/html40/entities/special.html；

http://code.stephenmorley.org/html-and-css/character-entity-references-cheat-sheet。

上面第一个链接中既包含 HTML 实体值，也包含实体需要作为::before 和::after 伪元素内容时的十六进制值。举个例子，如果你想在伪元素中添加十六进制值“（左双引号），则需要改写成如下形式：

```
e::before {content:"\201C";}
```

就是在数字前面加了一个反斜杠。相应地，右双引号的十六进制值改写后是\201D。

在 HTML 标记中，必须把所有和号（&）和小于号（<）都替换成相应的 HTML 实体，也就是&和<。因为“&”是实体的第一个字符，而“<”是 HTML 标签的第一个字符，它们都有特殊含义。如果不替换的话，浏览器一看到它们就会当成实体和标签来解释后面的内容。

第一步：设定字体和底层网格

这两段文本使用的主字体是 FontSquirrel 提供的字体 Crimson Roman。我下载了相应的字体包，放在了我 Web 服务器上（在编写代码的过程中，我在本地也保存了一份），然后把其中的@font-face 规则添加到了样式表中。之后，就可以在 font-family 属性中引用该字体了。

```
@font-face {
    font-family:'CrimsonRoman';
    src: url('fonts/Crimson-fontfacekit/Crimson-Roman-webfont.
        eot');
    src: url('fonts/Crimson-fontfacekit/Crimson-Roman-webfont.
        eot?#iefix') format('embedded-opentype'),
        url('fonts/Crimson-fontfacekit/Crimson-Roman-webfont.
            woff') format('woff'),
        url('fonts/Crimson-fontfacekit/Crimson-Roman-webfont.
            ttf') format('truetype'),
        url('fonts/Crimson-fontfacekit/Crimson-Roman-webfont.
            svg#CrimsonRoman') format('svg');
    font-weight: normal;
```

```
    font-style: normal;
}
* {margin:0; padding:0;}
body {
    font-family:"CrimsonRoman", georgia, times, serif;
    background-color:#fff;
    margin:0 10% 0;
    background-image:url(grid_24px.png);
}
```

按照我们的标准工作流程，这里先去掉了所有外边距和内边距，设定了主字体，又添加了左、右外边距，还为对齐文本临时添加了网格，结果如图 4-29 所示。

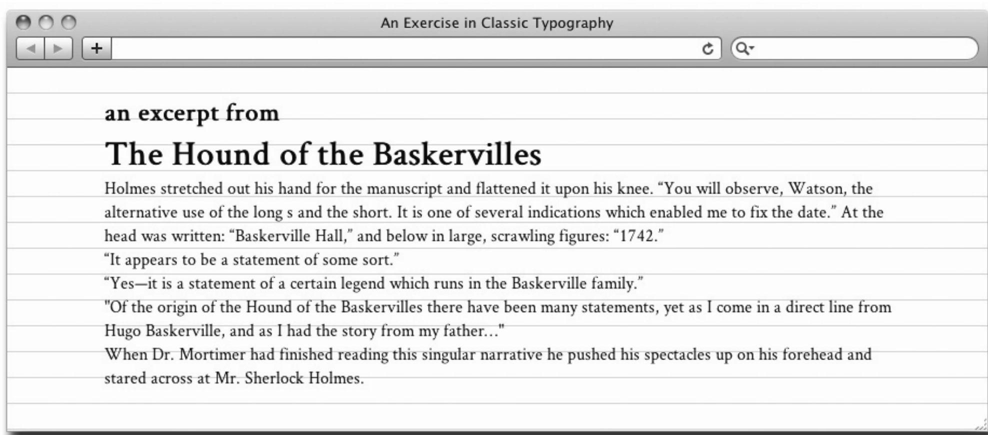


图 4-29　文本和网格已经就位，就等对齐了

第二步：设计标题

下面我们就从上到下，对齐页面中的每一个元素。第一行的副标题应该与第二行的主标题形成对比，因为主标题要使用手写字体，所以副标题就设定成间距较宽的小型大写字母。

```
body {
    font-family:"CrimsonRoman", georgia, times, serif;
    background-color:#fff;
    margin:29px 10% 0;
    background-image:url(grid_24px.png);
}
h2 {
```

```
    font-size:18px;
    line-height:24px;
    font-weight:bold;
    text-align:center;
    font-variant:small-caps;
    word-spacing:.5em;
    letter-spacing:.6em;
}
```

排版资源

http://ilovetypography.com 在这个网站上，你可以跟着设计师 John Boardley 一起深思，欣赏每一页中别具一格的排版。

http://www.thinkingwithtype.com 这是 *Thinking with Type* 一书的网站，作者 Ellen Lupton。网站中展示了很多漂亮又经典的版式，还包含一些字体及字体族的信息。

http://webtypography.net 这个站点自称为 The Elements of Typographic Style Applied to the Web—A practical guide to web typography。跟随网站的目录，可以找到各种常用的排版知识和技巧。

首先，我们用 `font-variant` 把副标题转换成小型大写字母，然后应用 `word-spacing` 和 `letter-spacing` 得到预期结果，如图 4-30 所示。



图 4–30　副标题不仅与网格对齐，而且也应用了字体和文本属性

接下来，打开 Google Web Fonts 网站，找到一种名为 Pinyon 的手写体，这种字体与主标题挺配的。然后，把 Google Web Fonts 生成的`<link>`标签复制粘贴到 HTML 文档的`<head>`标签里，这样就能在 `font-family` 声明中引用这种字体了。这里仍然需要用到图 4-25 中曾用过的"负/正外边距"技巧，把主标题移动到恰好与网格对齐，得到的效果如图 4-31 所示。

```
<link href='http://fonts.googleapis.com/css?family=Pinyon+Script'
    rel='stylesheet' type='text/css'>
h1 {
    font-size:60px;
```

```
    line-height:96px;
    font-family:"Pinyon Script", cursive;
    margin:4px 0 -4px;
    text-align:center;
    font-weight:normal;
    position:relative;
}
```



line-height 是网格宽度的 4 倍。



图 4-31 主标题已经应用了样式，并与网格对齐了

第三步：设计段落和引用

第一段的位置有点高，因此我们设定它的字体大小，以及——更重要的是设定它的行高。

```
p {font-size:18px; line-height:24px;}
```

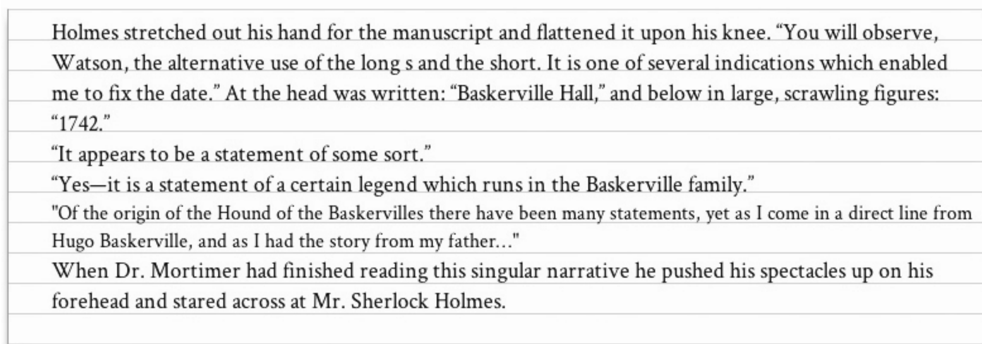
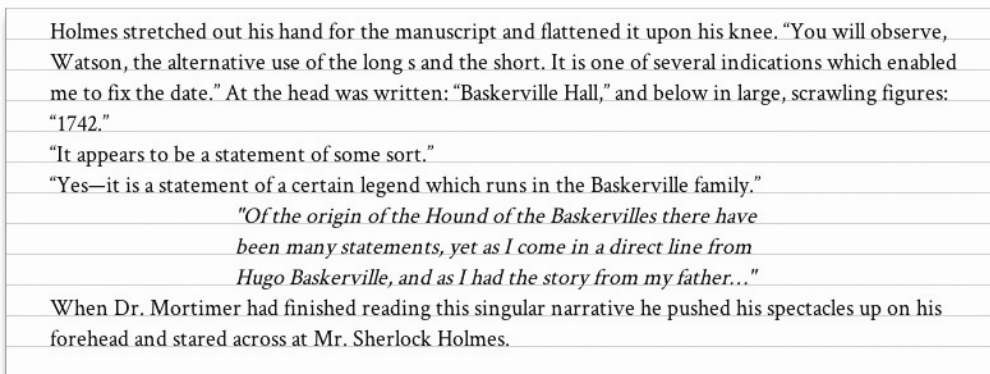


图 4-32 通过设定 line-height 让段落与网格对齐

前三段虽然对齐了，但后面的段落没有，这是因为第三段后面的引用（blockquote）也需要对齐。blockquote 中的文本包含在一个行内元素 q（quote）中，这个元素的默认样式是在文本开头和末尾加上引号。在此，我们来缩进 blockquote，但在其 q 子元素上设定字体大小和行高，因为它才是包含文本的元素：

```
blockquote {margin:0px 20%;}
q {font-size:18px; font-style:italic; line-height:24px;}
```


Holmes stretched out his hand for the manuscript and flattened it upon his knee. "You will observe, Watson, the alternative use of the long s and the short. It is one of several indications which enabled me to fix the date." At the head was written: "Baskerville Hall," and below in large, scrawling figures: "1742."

"It appears to be a statement of some sort."

"Yes—it is a statement of a certain legend which runs in the Baskerville family."

"Of the origin of the Hound of the Baskervilles there have been many statements, yet as I come in a direct line from Hugo Baskerville, and as I had the story from my father…"

When Dr. Mortimer had finished reading this singular narrative he pushed his spectacles up on his forehead and stared across at Mr. Sherlock Holmes.

图 4-33　引用与网格也对齐了

缩进引用并将文本变成斜体为页面增加了变化。另外，你看到了吗，在 blockquote 就位后，它后面的段落自然也就与网格对齐了。

第四步：首字下沉效果

现在，我们打算给第一段添加一个独特的首字下沉效果。所谓首字下沉，就是将段落开头的第一个字母变大，然后根据该字母与段落首行的位置关系，又可以分几种不同的风格。在这里，我们想让字母的顶部与段落首行顶部对齐，让字母的底部与段落第三行的基线对齐。

或许你以前也见过这种效果，而那些效果中的首字母都被包含在一个 span 元素里。这种做法对于从内容管理系统中动态取文本的网页并不适用。所以，我们这里要介绍一种不用修改标记的技巧。

为此，需要选择 h1 标题后面第一个段落的第一个字母，而组合使用紧邻同胞选择符+和::first-letter 伪元素可以做到。选择了第一个字母后，再增大字体并浮动它：

```
h1 + p::first-letter {
    font-family:times, serif;
    font-size: 90px;
    float:left;
    border:1px solid;
}
```

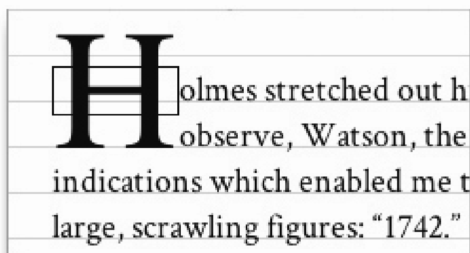


图 4-34　这里显示边框是为了看清楚首字母从段落文本中继承的 line-height 有多高

首字母增大了，但位置不对。由于我们实际上是想调整盒子的大小和位置，所以首字母的边框在此可以作为辅助。而边框告诉我这个盒子只大到了让两行文本绕排，原因是首字母继承了段落的行高和对齐方式。我们需要设定这个伪元素的 `line-height`，让它的盒子能包含首字母。



我们把 `line-height` 设定为小于 1 的值，以便元素盒子紧紧包住首字母，而不会强迫段落第四行也绕排。

```
h1 + p::first-letter {
    font-family:times, serif;
    font-size:90px;
    float:left;
    line-height:.65;
    border:1px solid;
}
```

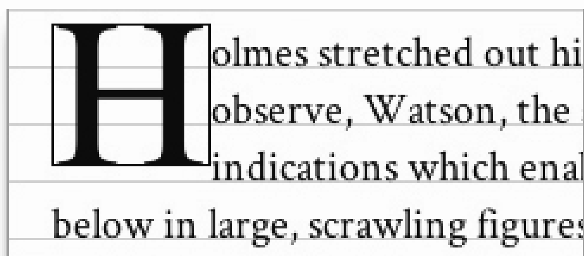


图 4-35　元素盒子紧紧包住首字母

如图 4-35 所示，增大盒子的 `line-height` 后，第三行也绕排了。现在，只要再给这个元素顶部添加一点内边距，就可以让字母的底部与段落第三行的基线对齐了。

```
h1 + p::first-letter {
    font-family:times, serif;
    font-size:-90px;
    float:left; line-height:.65;
    padding:.085em 3px 0 0;
}
```

我还添加了 3 像素的右内边距，以便首字母与段落之间空开一点距离。当然，边框也没用了，把它去掉之后的效果如图 4-36 所示。

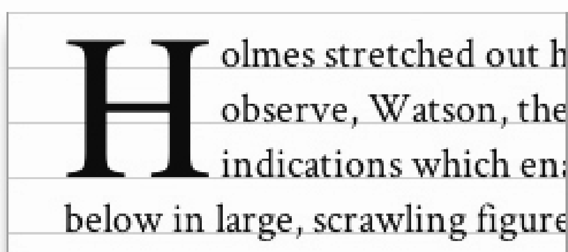


图 4-36　完成后的首字母下沉效果

第五步：设计第一行

首字母下沉效果做完了。可是，我希望大号首字母到小号正文之间有一个过渡，所以需要把段落第一行的文本变成小型大写字母。

```
h1 + p::first-line {
    font-variant:small-caps;
    letter-spacing:.15em;
}
```

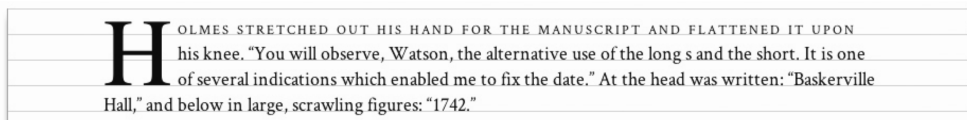


图 4-37　段落第一行变成了小型大写字母

这一次同样使用了紧邻同胞选择符，但组合的是`::first-line` 伪元素，实现了第一行字形的转换。使用伪元素而不是额外加标签的好处在于，当行的长度改变时，大写字母的效果会随之扩展。图 4-37 展示了这行大写字母在首字母与正文之间起到的衔接作用。

第六步：最后的修饰

段落之间没有足够的间距，让人很难看出它们的开头在哪里。为了与这本书的版式保持一致，我们不给段落之间加空白，而是缩进跟在其他段落后面的每一段，作为一组段落起头的一段不缩进。此外，引用内容两端的引号也太难看，所以要在`<q>`标签上应用`::before`和`::after`伪元素，插入 Crimson 字体的引号。最后，再从`body`元素中去掉网格背景（让它不再显示）。

```
/*只缩进跟在段落后面的段落*/
p + p {text-indent:14px;}
/*引用内容前面的引号*/
q::before {content:"\201C"}
/*引用内容后面的引号*/
q::after {content:"\201D"}
```

这几条声明都有必要解释一下。选择要缩进的段落使用的是紧邻同胞选择符，以保证只有段落后面的段落才会缩进。“When Dr. Mortimer...”开头的那一段，因为前面是一个`blockquote`，不是段落，所以就没有缩进，而且它也十分明显就是一个段落的开头（参见图 4-38）。

使用`::before`和`::after`伪元素给引用内容添加引号时使用的是十六进制值。在这里不能给`content`属性设定常规的 HTML 实体，只能使用十六进制实体值，而且只能是改写后的形式。关于 HTML 实体引用的详细介绍，请参考本节开头的那个附注栏。

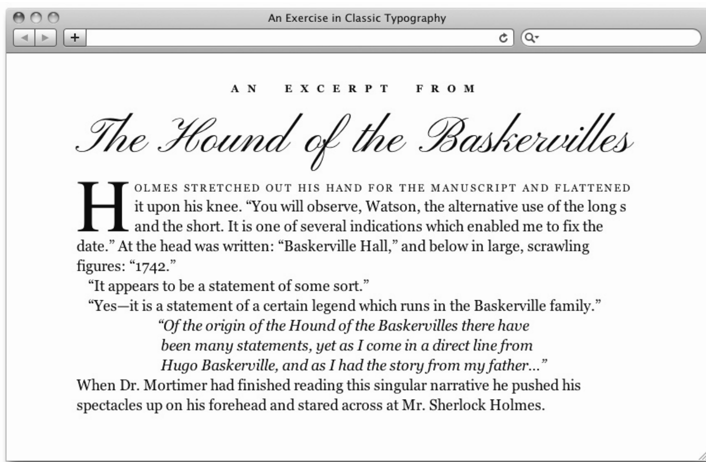


图 4–38 完成后的页面

经过最后的修饰，这个经典版式设计终于完成了。对于节选的那么几段文本来说，好像工作量还不小。不过别忘了，这些样式其实是很容易应用给整整一本书的。

4.5　小结

本章，我们介绍了与字体和文本相关的很多 CSS 属性，学习了为页面设定字体的几种不同方式，也讨论了为文本应用样式的各种技巧。下一章，我们就在文字排版的基础上更上一层楼，学习设计多栏页面布局的技术。

5

页面布局

本章，我们来学习多栏页面布局。很多网站为了在第一屏尽可能多地显示信息，都采用了多栏布局。这里所说的“第一屏”就是用户无须滚动就能看到的页面区域，相当于传统报纸行业所说的“折痕之上”的版面。一般来说，两栏、三栏，甚至四栏布局都很常见。但无论几栏，每个页面都要涉及一些关键技术和指导思想，这些内容本章都将介绍。

这一章会介绍实现多栏布局的几种方法。主要介绍“用内部 DIV 创建浮动的栏”，这是多年来一直被业界用于创建多栏布局的经典技术。另外，随着支持 CSS3 的浏览器比例越来越高，一些新的布局技术也可以用在大型站点中了，本章当然不会忽略这部分内容。今天，我们不仅可以使用 `box-sizing` 属性代替内部 DIV，也可以使用让元素行为跟表格一样的 CSS3 的 `display` 属性（却不必向 HTML 中添加表格）。因此，创建流动的全长栏也变得简单多了。

我会把上述经典技术和新技术的优缺点都摆出来，还会告诉大家对那些老版本浏览器（当然主要指 IE8 及更早版本），有什么后备手段和“腻子脚本”（polyfill）可以使用。这些都是你选择技术时需要考虑的。好吧，闲话少说，就从布局的基本概念开始讲吧。

5.1　布局的基本概念

多栏布局有三种基本的实现方案：固定宽度、流动、弹性。

- 固定宽度布局的大小不会随用户调整浏览器窗口大小而变化，一般是 900 到 1100 像素宽。其中 960 像素是最常见的，因为这个宽度适合所有现代显示器，而且能够被 16、12、10、8、6、5、4 和 3 整除，不仅容易计算等宽分栏的数量，而且计算结果也能得到没有小数的像素数。



流行的 CSS 布局框架 960 Grid（http://www.960.gs），就是基于 960 像素宽的网格创建的。

- 流动布局的大小会随用户调整浏览器窗口大小而变化。这种布局能够更好地适应大屏幕，但同时也意味着放弃对页面某些方面的控制，比如随着页面宽度变化，文本行的长度和页面元素之间的位置关系都可能变化。Amazon.com 的页面采用的就是流动中栏布局，在各栏宽度加大时通过为内容元素周围添加空白来保持内容居中，而且现在的导航条会在布局变窄到某个宽度时收缩进一个下拉菜单中，从而为内容腾出空间。
- 今天，越来越多的浏览器都支持 CSS 媒体查询了。这就让基于浏览器窗口宽度提供不同的 CSS 样式成为可能。在这种形势下，适应各种屏幕宽度的可变固定布局，正逐步取代流动布局。这种可变的固定布局能够适应最大和最小的屏幕，业界称之为响应式设计。本书第 8 章将专门介绍响应式设计相关的 CSS 技术。
- 弹性布局与流动布局类似，在浏览器窗口变宽时，不仅布局变宽，而且所有内容元素的大小也会变化，让人产生一种所有东西都变大了的感觉。到目前为止，我还没见过设计得非常好的弹性布局，主要是因为它太过复杂了。本章不介绍这种布局，而只把笔墨花在固定宽度布局和流动布局上。

下面，我们先来看一看页面的高度和宽度有什么区别。

布局高度与布局宽度

在实际地创建页面布局之前，我想先说说应该怎么看待布局的高度和宽度，因为这两者的控制方法实在太不一样了。

布局高度

多数情况下，布局中结构化元素（乃至任何元素）的高度是不必设定的。事实上，我甚至想告诉你根本不应该给元素设定高度。除非你确实需要这样做，比如在页面中创造一个绝对定位的元素。

为什么正常情况下都应该保持元素 height 属性的默认值 auto 不变呢？很简单，只有这样元素才能随自己包含内容的增加而在垂直方向上扩展。这样扩展的元素会把下方的元素向下推，而布局也能随着内容数量的增减而垂直伸缩。假如你明确设定了元素的高度，那么超出的内容要么被剪掉，要么会跑到容器之外——取决于元素 overflow 属性的设定。

布局宽度

与高度不同，我们需要更精细地控制布局宽度，以便随着浏览器窗口宽度的合理变化，布局能够作出适当的调整，确保文本行不会过长或过短。如果随意给元素添加内边距、边框，或者元素本身过大，导致浮动元素的宽度超过包含元素的布局宽度，那浮动元素就可能“躲”到其他元素下方。

当然啦，即使必须设定栏宽，也不要给包含在其中的内容元素设定宽度，应该让这些内容元素自动扩展到填满栏的宽度。本书前面已经讲过了，这是块级元素的默认行为。简言之，就是让栏宽限制其中内容元素的宽度。

以上这些细节在实际地创建布局时会更容易明白。眼下，只要知道应该控制布局宽度，而让内容决定布局高度这个一般原则就好了。

5.2　三栏-固定宽度布局

本节，我来教你创建三栏布局。只要掌握了创建三栏布局的技术，你想搞多少栏就能搞多少栏。由于本章主要关注页面结构，为了让大家直观地看到整个过程到底发生了什么，咱们要给每个栏加上深浅不同的背景色。

就从一个简单的居中的单栏固定布局开始吧。为了让布局好看一点，我写了一些简单的样式。为节省版面，我们没有给出这些样式，实际上它们对本节要讲的技术没有任何影响。主要标记的 ID 是 wrapper（外包装），这个容器里包含了一栏。

```
<div id="wrapper">
    <article>
        <! -- 这里是一些文本元素 -->
    </article>
</div>
```

布局相关的 CSS 如下：

```
#wrapper {width:960px; margin:0 auto; border:1px solid;}
article {background:#ffed53;}
```

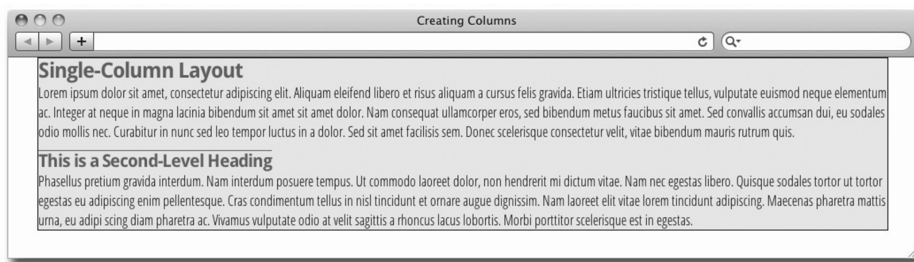


图 5-1　为 article 添加背景色和边框，以看清这个居中的栏

如图 5-1 所示，通过给整个外包装设定宽度值，并将其水平外边距设定为 auto，这个单栏布局在页面上居中了。随着向里添加内容，这一栏的高度会相应增加。外包装中的 article 元素本质上就是一个没有宽度的块级盒子（关于“没有宽度的盒子”，请参见 3.2 节），它水平扩展填满了外包装。下面，我们再向外包装里添加一个导航元素，让它作为第二栏。

```
<div id="wrapper">
    <nav>
        <!-- 无序列表 -->
    </nav>
    <article>
        <! -- 文本 -->
    </article>
</div>
```

我们得浮动作为两栏的容器元素，让它们并排显示，这是我们在 3.3 节讲过的。

```
#wrapper {width:960px; margin:0 auto; border:1px solid;}

nav {
    width:150px;
```

```
    float:left;
}
nav li {
    /*去掉列表项目符号*/
    list-style-type:none;
}
article {
    width:810px;
    float:left;
    background:#ffed53;
}
```

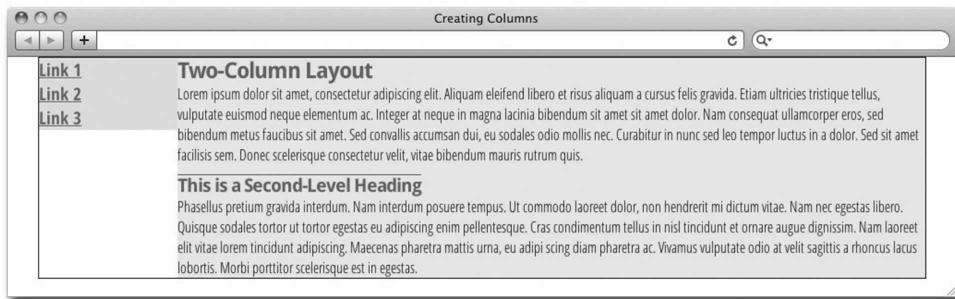


图 5-2 在布局中添加了第二栏

如图 5-2 所示，把两栏容器元素的总宽度设定为外包装的宽度（150 + 810 = 960），并浮动它们，就可以创造出并肩排列的两栏来。每一栏的长度取决于内容多少。采用同样的方法，可以添加第三栏（或任意多个栏）。



要了解如何让所有栏看起来都同布局一样高，请参考 5.3.1 节介绍的“人造栏技术”。

```
<div id="wrapper">
    <nav>
        <!-- 无序列表 -->
    </nav>
    <article>
        <! -- 文本 -->
    </article>
    <aside>
        <! -- 文本 -->
    </aside>
</div>
```

接下来，我们要调整 article 这一栏的宽度，为第三栏腾出空间。

```
#wrapper {width:960px; margin:0 auto; border:1px solid;}
nav {
    width:150px;
    float:left;
    background:#dcd9c0;
}
nav li {
    list-style-type:none;
}
article {
    width:600px;
    float:left;
    background:#ffed53;
}
aside {
    width:210px;
    float:left;
    background:#3f7ccf;
}
```

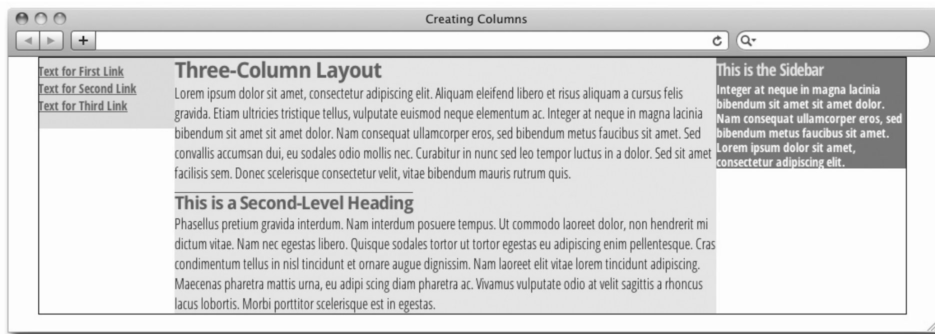


图 5–3　布局中有了三个浮动栏

如图 5-3 所示，通过把三个浮动容器的总宽度设定为恰好等于外包装的宽度（150 + 600 + 210 = 960），就有了三栏布局的框架。就用这种办法，我可以想加多少栏就加多少栏，只要它们的总宽度等于外包装的宽度即可。当然，多栏布局通常都有与布局同宽的页眉和页脚，下面我们就来添加。

```
<div id="wrapper">
    <header>
```

```
        <!-- 标题 -->
    </header>
    <nav>
        <!-- 无序列表 -->
    </nav>
    <article>
        <! -- 文本 -->
    </article>
    <aside>
        <! -- 文本 -->
    </aside>
    <footer>
        <!-- 文本 -->
    </footer>
</div>
```

我们希望页眉和页脚与布局同宽，而且它们默认就与布局同宽，所以如图 5-4 所示，在此我们只简单地为它们设定了背景色，以便能看到它们在哪儿。

```
header {background:#f00;}
footer {background:#000;}
```

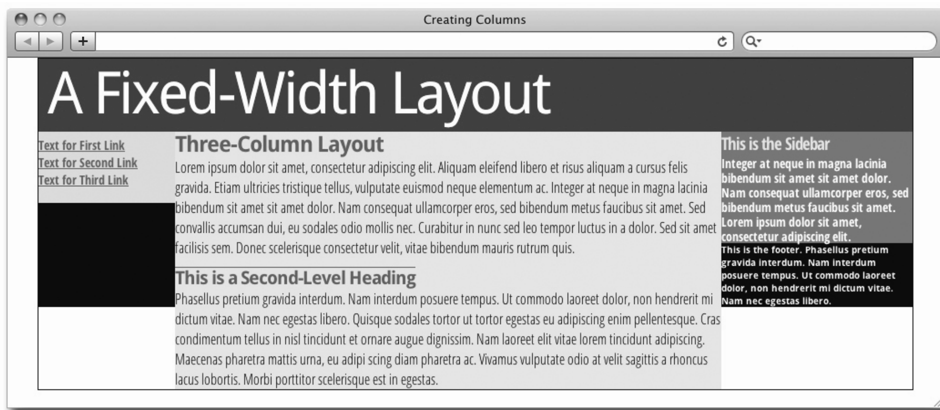


图 5–4　页眉还不错，但页脚却移动到了浮动栏的后面

此时的页眉与布局同宽，其内容高度也比较合适。但是，页脚位于浮动元素后面，所以就会尽量往上移动。解决这个问题也很简单，代码如下，效果如图 5-5 所示。

```
footer {clear:both;}
```

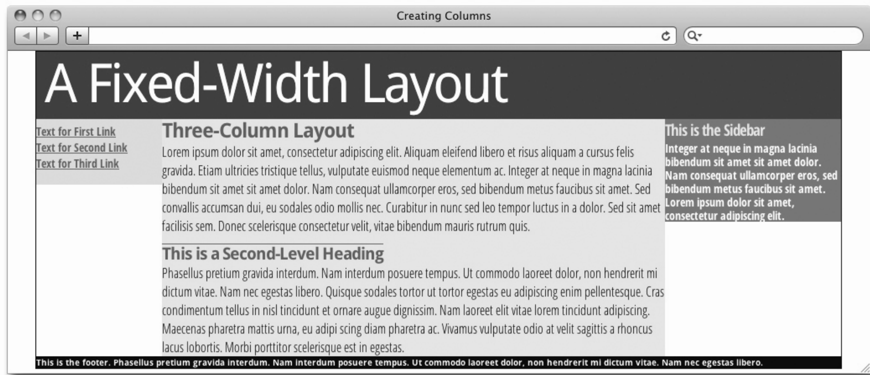


图 5–5　应用清除之后，页脚把自己定位到了最长一栏的下方

为页脚应用 clear:both（clear:left 效果也一样，因为这里只有左浮动元素），就可以阻止它向上移动，不让它超过浮动元素的下方边界。这么一条简单的规则，就可以保证页脚始终都位于最长栏的下方。

到目前为止，我们为 HTML 标记应用的 CSS 如下（没多少文本样式）。

```
* {margin:0; padding:0;}
#wrapper {width:960px; margin:0 auto; border:1px solid;}
header {background:#f00;}
nav {
    width:150px;
    float:left;
    background:#dcd9c0;
}
nav li {
    list-style-type:none;
}
article {
    width:600px;
    float:left;
    background:#ffed53;
}
aside {
    width:210px;
    float:left;
    background:#3f7ccf;
}
footer {clear:both; background:#000;}
```

接下来，我们主要解决布局中的两个主要问题。首先，内容与各栏边界紧挨在一起，太拥挤；其次，每栏高度由文本多少决定，而如果每栏都与布局一样高则更好。我们先来为内容周围添加一些空白。别以为这是件简单的事儿，一会儿你就知道了。

为栏设定内边距和边框

只要一调整各栏中的内容，布局就可能超过容器宽度，而右边的栏就可能滑到左边的栏下方。一般来说，两种情况下可能会发生这种问题。

- 为了让内容与栏边界空开距离，为栏添加水平外边距和内边距，或者为了增加栏间距，为栏添加外边距（只要开始给布局添加样式，就一定会采用这里说的一种做法，甚至双管齐下），导致布局宽度增大，进而浮动栏下滑。换句话说，右边浮动的栏因为没有足够的空间与其他栏并列，就会滑到左边栏的下方。
- 在栏中添加大图片，或者没有空格的长字符串（如长 URL），也会导致栏宽超过布局宽度。同样，这种情况下右边的栏也会滑到左边的栏下方。

好了，下面我们就来试试添加内边距，增大内容与栏边界的距离。这次从中间一栏开始。

```
article {
    width:600px;
    float:left;
    background:#ffed53;
    padding:10px 20px;
}
```

如图 5-6 所示，中间栏的文本四周都增加了间距，与栏边界保持了一定距离。可是，这样一来由于中间栏的总宽度增加，导致右边的栏不能再与前两栏并排在一起，结果就跑到了左边栏的下方。还记得吗，第 3 章在讲盒模型的时候我们说过，为固定宽度的元素添加水平外边距、边框和内边距，会导致元素盒子变宽。像这样增大浮动栏的宽度，几乎总会造成图 5-6 所示的“浮动滑移”问题。好在，我们也有三种方法来预防该问题发生。



图 5-6　中间栏中的内容与栏边界之间有了空间，但由于这个栏占据的空间增大，导致右边的栏滑到了左边的栏下方

- 从设定的元素宽度中减去添加的水平外边距、边框和内边距的宽度和。
- 在容器内部的元素上添加内边距或外边距。
- 使用 CSS3 的 box-sizing 属性切换盒子缩放方式，比如 section {box-sizing: border-box;}。应用 box-sizing 属性后，给 section 添加边框和内边距都不会增大盒子，相反会导致内容变窄。

我们分别试验一下这几种手段。

1. 重设宽度以抵消内边距和边框

比如我们给 600 像素宽的栏又添加了 20 像素的内边距，为了抵消增加的内边距，可以把栏宽减少 40 像素而设定为 560 像素。这样，右边的栏就能归位。问题在于，每次只要调整内、外边距就要重设布局宽度，有点烦人。因此这个办法虽然可行，却不够理想。说不定哪一回调整内边距或边框就会导致布局错乱。

2. 给容器内部的元素应用内边距和边框

把外边距和内边距应用到内容元素上确实奏效。前提是这些元素没有明确地设定宽度，这样它们的内容才会随着内、外边距的增加而缩小。就像盒模型定义所说的，没有宽度的元素在水平方向上会适应其父元素，其内容会随着外边距、边框和内边距的增加而减少。

然而，一栏之中可能会包含大量不同内容的元素。假如将来又决定调整内容与容器边界的距离，就必须每个元素都要进行调整，这样不仅麻烦，而且容易出错。况且，给栏添加边框同样会增大栏宽，不可能通过为其包含的内容元素逐个应用样式来做到。

所以说，与其为容器中的元素添加外边距，不如在栏中再添加一个没有宽度的 div，让它包含所有内容元素，然后再给这个 div 应用边框和内边距。如此一来，只要为内部 div 设定一次样式，就可以把让所有内容元素与栏边界保持一致的距离。而且，将来再需要调整时也会很方便。任何新增内容元素的宽度都由这个内部 div 决定。

采用这种方法除了标记中多了一个 div 元素外，唯一的问题就是那些反对把标记用于表现用途的纯粹论者会跟你叫嚣。关于我对这个问题的看法，请参考附注栏“关于表现性标记的思考”，另外一个附注栏“子-星选择符”也给出了用代码替换内部 div 的方案。

关于表现性标记的思考

HTML 的目的是语义，也就是给内容赋予含义。而 CSS 呢，是为了把表现性的样式分离出来才发明的。不过，有些表现性标记是有害的，而有些则没有副作用。使用表格来创建多栏布局，或者使用
标签在段间换行，却不使用<p>标签，这种做法的确不值得提倡，因为这会造成内容难以移植。比如说吧，用三个表格单元作为三栏，这种布局到哪都会显示成表格，就算是在完全不合适的智能手机里也一样。如果表现性标记无法用 CSS 修改，或者在 CSS 不可用时也要迫使用户接受，那就是滥用 HTML。可是，div 或 span 这种中性的元素，对默认样式没有影响，除非你给它们应用样式，否则它们就跟不存在一样。所以，我认为添加这种元素达到表现性的目的是完全可以接受的。

下面我就为大家示范怎么用内部 div 修复图 5-6 中存在的问题。

```
<article>
    <div class="inner">
        <!-- 文本 -->
    </div>
</article>
```

现在，把造成问题的内边距从栏上去掉。为了示范这个技术有多好用，接下来我们不仅要给内部 div 应用内边距，还要给它应用外边距和边框。

```
article {
    width:600px;
    float:left;
    padding:10px 20px;
    background:#ffed53;
}

article .inner {
    margin:10px;
    border:2px solid red;
    padding:20px;
}
```

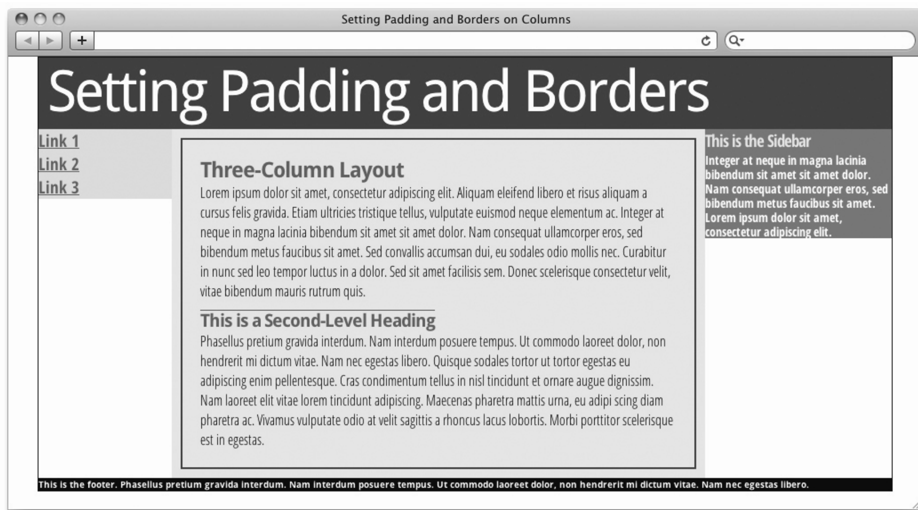


图 5-7　给没有宽度的内部 div 应用外边距、边框和内边距后，中间栏的宽度没有变化，右边的栏仍然还在原来的位置上

如图 5-7 所示，中间栏的宽度并未因此有什么变化，因为内容区减少的宽度抵消了应用到内部 div 上的外边距、边框和内边距的总宽度。总之，由此可以得出一个结论：如果布局中的栏是浮动的，而且都设定了宽度，你就根本不要去动它！要动，就把内容放在内部 div 里，动这个 div。

好了，解决问题的关键已经讲清楚了。接下来我们去掉中间栏的外边距、边框和内边距，给其他两栏也添加内部 div，然后只给这三栏加上内边距。

```
<div id="wrapper">
    <header>
```

```
        <!-- header text -->
    </header>
    <nav>
        <div class="inner">
            <ul>
                <!-- 链接 -->
            </ul>
        </div>
    </nav>
    <article>
        <div class="inner">
            <!-- 文本 -->
        </div>
    </article>
    <aside>
        <div class="inner">
            <!-- 文本 -->
        </div>
    </aside>
    <footer>
        <!-- 文本 -->
    </footer>
</div>
```

接下来我们就利用这个 div 为三个栏中的内容创造间距。此外，还居中了页脚中的内容。

```
nav .inner {padding:10px;}
article .inner {padding:10px 20px;}
aside .inner {padding:10px;}
footer {text-align:center}
```

如图 5-8 所示，三栏文本与栏边界之间有了内边距生成的必要距离，而 footer 元素中的文本也居中了。

以上措施使图 5-5 所示的布局有了明显改观。就这么简单的几下，布局就显得更专业了。处理栏及其内部 div 的关键在于，浮动栏并设定栏宽，但不给任何内容元素设定宽度。要让内容元素扩展以填充它们的父元素——内部 div。这样，只要简单地设定内部 div 的外边距和内边距，就可以让它们以及它们包含的内容与栏边界保持一定距离。

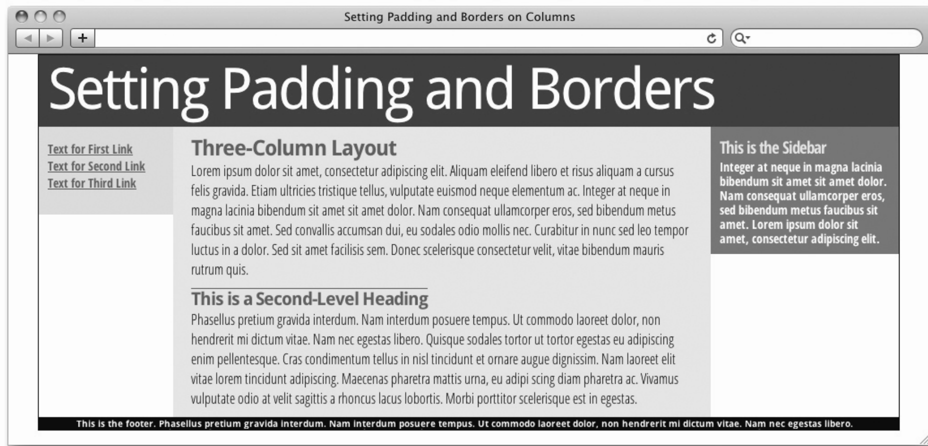


图 5-8　通过为内部 div 应用内边距，布局的宽度并没有改变

注意，如果容器的上、下边框不可见，内部 div 的上、下外边距会叠加。要是你遇到了这个问题，可以只为容器设定垂直内边距。但要小心一点，别一块儿也添加水平内边距，比如 article {padding:20px 0;}，就只设定了上、下内边距。

子-星选择符

所谓“子-星选择符”就是一个组合选择符，利用它可以不使用内部 div 就能设定一栏中所有元素的外边距。

星号选择符可以选择“所有元素”，故而，在一个选择符后面加个星号，比如 someSelector * 就可以选择 someSelector 所代表元素的所有后代元素。子选择符可以选择“某元素的子元素”，故而，把子选择符放到星号前面，比如 someSelector > *就会只选择 someSelector 所代表元素的所有子元素，而非后代元素。这正好适用于选择容器内部的所有顶部元素，然后设定它们的外边距。比如，对于 section 栏，设定 section > * {margin:0 10px;}，就能为栏中所有子元素，不包括其他后代元素，各应用 10 像素的左、右外边距。 使用“子-星选择符”要注意两点。第一，在为子元素设定垂直外边距时，只能使用 margin-top 和 margin-bottom，不能使用简写的 margin，否则会抵消用“子-星选择符”应用给这些元素的水平外边距。如果你想进一步缩进某个子元素的内容，就应该给该子元素应用内边距。

第二，“子-星选择符”有潜在性能问题，因为它会导致浏览器遍历整个 DOM 结构去查找所有匹配的元素。但我也发现这一点性能影响几乎可以忽略不计。假如你的页面真的包含几千上万个元素，那倒确实该考虑用 ySlow 或其他性能度量工具测一测这个选择符的影响。

3. 使用 box-sizing:border-box

这是最简单的一个办法。只要在三个浮动的栏的 CSS 规则中分别加上 box-sizing:border-box 声明，再给栏添加内边距（和边框）就不会导致盒子的宽度变化了。此时，既不用调整栏宽去抵消增加的内边距，也不用使用内部 div。添加内边距的结果就是内容收缩。以下就是简洁清晰的没有内部 div 的标记。

```
<div id="wrapper">
    <header>
        <!-- 标题 -->
    </header>
    <nav>
        <ul>
            <!-- 链接 -->
        </ul>
    </nav>
    <article>
        <!-- 文本 -->
    </article>
    <aside>
        <!-- 文本 -->
    </aside>
    <footer>
        <!-- 文本 -->
    </footer>
</div>
```

而以下就是 CSS 规则。

```
* {margin:0; padding:0;}
#wrapper {width:960px; margin:0 auto; border:1px solid #000;
    overflow:hidden;}

header {background:#f00;}
nav {
    box-sizing:border-box;
    width:150px;
    float:left;
    background:#dcd9c0;
    padding:10px 10px;
}
```

```
/*去掉列表项目符号*/
nav li {list-style-type:none;}
article {
    box-sizing:border-box;
    width:600px;
    float:left;
    background:#ffed53;
    padding:10px 20px;
}
aside {
    box-sizing:border-box;
    width:210px;
    float:left;
    background:#3f7ccf;
    padding:10px 10px;
}
footer {clear:both; background:#000;}
```

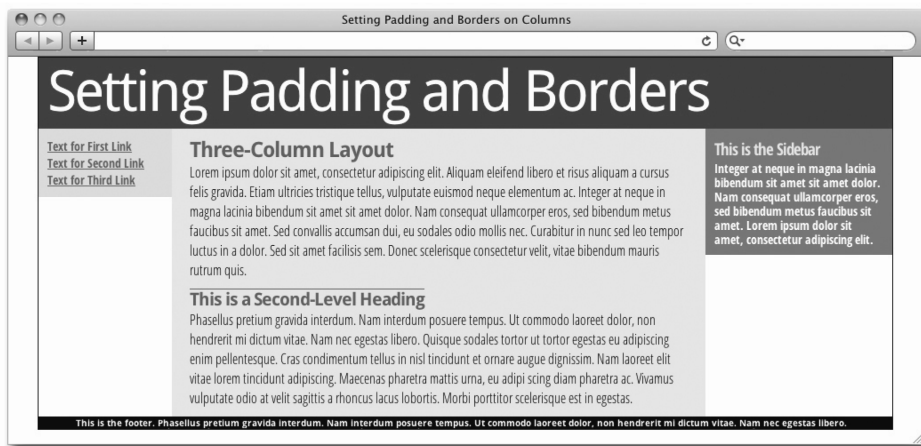


图 5–9　使用 box-sizing 属性后，可以直接给栏应用内边距

如图 5-9 所示，直接给栏应用内边距会导致内容变窄，但不会影响布局。听起来容易的办法总会有一个“但是”，这里的“但是”要说的是 IE6 和 IE7 不支持 `box-sizing` 属性。不过，有一个专门解决这个问题的腻子脚本（polyfill），名叫 borderBoxModel.js。你可以使用条件注释（以便只有 IE6 和 IE7 加载）把它添加到 HTML 标记之后、结束的`</body>`标签之前，以保证在加载 DOM 之后再执行该脚本：



本书附录介绍了什么是腻子脚本，以及为什么需要它们。

```
<body>
<!-- HTML 标记 -->
<!-- 只让 IE8 之前的 IE 加载它 -->
<!--[if lt IE 8 ]>
<script src="helpers/borderBoxModel.js"></script>
<![endif]-->
</body>
```



最新版本的 borderBoxModel.js 腻子脚本以及它的用途和局限性，可以参考这里：https://github.com/albertogasparin/borderBoxModel。另外，也可以读一读本人的这篇谈 box-sizing 的文章：http://albertogasparin.it/articles/2012/02/start-using-css-box-sizing-today。

这样，IE6 和 IE7 就可以根据 box-sizing 属性的设定正确地调整栏的大小了。在依靠内部 div 几年之后，经过试验我发现，不仅给浮动的栏，甚至给所有元素都应用这个不同的盒缩放模型都是没有问题的，我在 CSS 里会添加这一条规则：

```
* {box-sizing:border-box}
```

如此一来，页面中的盒模型就全都符合逻辑了。换句话说，每个盒子的宽度并不是内容区的宽度，而是一经设定就不可变的真正的盒子宽度。在此期间，我经常关注一些 Web 大牛，比如 Paul Irish。建议大家读一读他的相关文章 http://paulirish.com/2012/box-sizing-border-box-ftw。另外，别忘了看看文章底下的评论，因为有人提出了一些疑问，也不是所有人都同意他的观点。

好了，下一节我们就讲一讲实现流动布局的几种方式。

预防过大的元素

设计一个将来可能由他人维护的动态网站时，需要考虑得更长远一些。比如，应该预见到可能出现一些过大的元素。如果将来有一张比浮动栏更宽的图片被放到栏中，就会导致布局变宽，而右边的栏又会滑到下方。为此，一个简单的预防措施就是添加一条 `.inner img {max-width:100%;}` 声明，以便限制图片的宽度不超过其父元素（在此就是内部 div）。

另一个办法是给每个栏（或者内部 div，如果你用了的话）添加 overflow:hidden 声明。这条声明不会缩小图片以适应父元素，而会将它（以及任何过大元素）超出容器边界的部分剪切掉。

> 动态网站中另一个潜在的问题是换行。HTML 只会在单词间空格的地方换行。一些长 URL，甚至一些长单词，在栏比较窄的情况下，都会导致栏宽过大。因此，还应该给所有栏的外包装元素应用 word-wrap:break-word 声明，以便所有栏及其内容继承这个设定。有了这条声明，浏览器会把过长的词断开显示在不同行上。只是 word-wrap 没有定义在哪里断开，因此结果完全是随机的，而且没有连字符。不过，这条规则只在需要时才会起作用，而且能保护布局不会被长 URL 顶得支离破碎。建议你在每一栏中都用长 URL、大图片，以及包含内容过多的元素测试一下布局，看看这些声明到底会不会起作用，并发现更多需要事先考虑保护措施的其他漏洞。

5.3　三栏–中栏流动布局

实现中栏流动布局有两种方法。一种是在中栏改变大小时使用负外边距定位右栏，另一种是使用 CSS3 让栏容器具有类似表格单元的行为。负外边距适合比较老的浏览器，而 CSS 的 table 属性则要简单得多。本节介绍这两种方法。

5.3.1　用负外边距实现

实现三栏布局且让中栏内容区流动（不固定）的核心问题，就是处理右栏的定位，并在中栏内容区大小改变时控制右栏与布局的关系。

Web 开发人员 Ryan Brill 给出的解决方案是控制两个外包装（通过 ID 值为 wrapper）容器的外边距。其中一个外包装包围所有三栏，另一个外包装只包围左栏和中栏。由于相应的标记和 CSS 与我们前面刚讲过的固定栏宽布局相似，所以有些细节就没有必要再重复了。不过，以下代码加粗了相应的外包装元素及 CSS 规则，并配上了简明的注释，最后还给出了两个屏幕截图（图 5-10a 和图 5-10b）。

```
<div id="main_wrapper">
    <header>
        <!-- 页眉-->
    </header>
    <div id="threecolwrap">/*三栏外包装（包围全部三栏）*/
        <div id="twocolwrap">/*两栏外包装（包围左栏和中栏）*/
            /*左栏*/
            <nav>
                <!-- 导航 -->
```

```
            </nav>
            /*中栏*/
            <article>
                <!-- 区块 -->
            </article>
        </div>/*结束两栏外包装（twocolwrap）*/
        /*右栏*/
        <aside>
            <!-- 侧栏 -->
        </aside>
    </div>/*结束三栏外包装（threecolwrap）*/
    <footer>
        <!-- 页脚 -->
    </footer>
</div>
```

以下就是 CSS 规则。



下载的代码中包含一些与修复 IE6 和 IE7 的 bug 有关的代码，这些代码都有注释。

```
* {margin:0; padding:0;}
body {font:1em helvetica, arial, sans-serif;}
div#main_wrapper{
    min-width:600px; max-width:1100px;
    /*超过最大宽度时，居中布局*/
    margin:0 auto;
    /*背景图片默认从左上角开始拼接*/
    background:url(images/bg_tile_150pxw.png) repeat-y #eee;
}
header {
    padding:5px 10px;
    background:#3f7ccf;
}
div#threecolwrap {
    /*浮动强制它包围浮动的栏*/
    float:left;
    width:100%;
    /*背景图片右对齐*/
    background:url(images/bg_tile_210pxw.png) top right repeat-y;
}
div#twocolwrap {
    /*浮动强制它包围浮动的栏*/
```

```
    float:left;
    width:100%;
    /*把右栏拉到区块外边距腾出的位置上*/
    margin-right:-210px;
}
nav {
    float:left;
    width:150px;
    background:#f00;
    padding:20px 0;
}
/*让子元素与栏边界保持一定距离*/
nav > * {margin:0 10px;}
article {
    width:auto;
    margin-left:150px;
    /*在流动居中的栏右侧腾出空间*/
    margin-right:210px;
    background:#eee;
    padding:20px 0;
}
*让子元素与栏边界保持一定距离*/
article > * {margin:0 20px;}
aside {
    float:left;
    width:210px;
    background:#ffed53;
    padding:20px 0;
}
*让子元素与栏边界保持一定距离*/
aside > * {margin:0 10px;}
footer {
    clear:both;
    width:100%;
    text-align:center;
    background:#000;
}
```

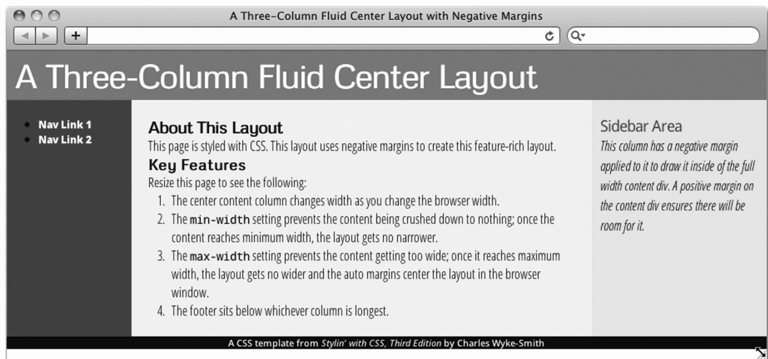


图 5-10a　屏幕变宽时，中栏变宽，左栏和右栏宽度不变

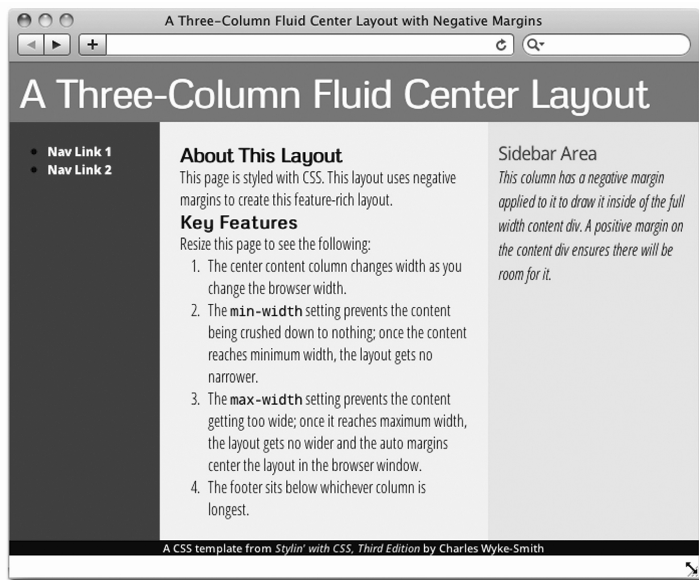


图 5-10b　屏幕变窄时，中栏变窄，左栏和右栏宽度不变

图 5-10a 和图 5-10b 展示了流动中栏布局。下面简单说明其原理。三栏中的右栏是 210 像素宽。为了给右栏腾出空间，中栏 `article` 元素有一个 210 像素的右外边距。当然，光有这个外边距只能把右栏再向右推 210 像素。别急，包围左栏和中栏的两栏外包装上 210 像素的负右外边距，会把右栏拉回 `article` 元素右外边距（在两栏外包装内部右侧）创造的空间内。中栏 `aticle` 元素的宽度是 `auto`（原文 100%有错误。——译者注），因此它仍然会力求占据浮动左栏剩余的所有空间。可是，一方面它自己的右外边距在两栏外包装内为右栏腾出了空间，另一方面两栏外包装的负右外边距又把右栏拉到了该空间内。总之，这是个很巧妙的设计。

人造栏技术

有人可能会纳闷，这些栏怎么都跟布局一样高呢？实话跟你说吧，你看到的都是假象！这里我采用了一种叫"人造栏"的技术，这样才让所有栏看起来都一样高了。这个技术说来也简单，就是给包围栏的外包装元素应用与各栏同宽的背景图片和背景色。外包装元素跟它们包含的栏不一样，它们的高度就是布局高度，当然与内容区的高度相同。通过在它们的背景的垂直方向上重复拼接背景图片，就可以在视觉上造成各栏与布局同高的假象。这个例子分别为左栏和右栏应用了不同的背景图片（如图 5-11 所示），为流动的中栏应用了背景色。



图 5-11　为流动中栏布局的左栏和右栏准备的两张背景图片

如前面 CSS 加粗的代码所示，左栏的背景图片加在了 `div#main_wrapper` 上。右栏的背景图片加在了 `div#threecolwrap` 上，而且让它沿该 `div` 的右侧垂直拼接。中栏的背景色也加在了 `div#main_wrapper` 上，这个背景色实际上是整个布局的背景色。而位于两侧的两栏的背景图片，以及页眉和页脚的背景色，都会覆盖这个背景色（子元素覆盖父元素）。因此，只能在中栏看到该全局背景色。



访问本书网站 http://www.stylinwithcss.com，可以免费阅读使用 jQuery 和渐变图生成人造栏的一章。

在这个页面布局的例子中，还有一个知识点值得注意。那就是我使用"子–星选择符"为内容元素添加了水平外边距（如 `nav > * {margin:0 10px;}`），而没有采用给内部 `div` 添加内边距的方法。这样就让你在实际案例中看到该选择符的应用。要了解"子–星选择符"的细节，请参考 5.2 节中"为栏设定内边距和边框"小节。

5.3.2　用CSS3 单元格实现

尽管利用 HTML 的`<table>`标签实现多栏布局是难以接受的，但使用 CSS 让布局形如表格则是绝对可以接受的。这种方法不会导致固定不变的表格布局，也不会出现难以重新应用样式的问题（比如在手持设备上表现为一栏）。说到创建布局，表格的行为确实是非常符合要求的，下面我来解释一下。

在最简单的情况下，表格由三个元素构成。一个表格外包装`<table>`，包含着表格行

<tr>和表格数据<td>，比如下面这个例子。

```
<table>
    <tr> <td>Cell 1</td><td>Cell 2</td><td>Cell 3</td> </tr>
    <tr> <td>Cell 1</td><td>Cell 2</td><td>Cell 3</td> </tr>
</table>
```

我们知道，CSS 可以把一个 HTML 元素的 `display` 属性设定为 `table`、`table-row` 和 `table-cell`。通过这种方法可以模拟相应 HTML 元素的行为。而通过 CSS 把布局中的栏设定为 `table-cell` 有三个好处。

- 单元格（table-cell）不需要浮动就可以并排显示，而且直接为它们应用内边距也不会破坏布局。
- 默认情况下，一行中的所有单元格高度相同，因而也不需要人造的等高栏效果了。
- 任何没有明确设定宽度的栏都是流动的。

这三个好处解决了本章前面学习布局时遇到的问题。然而，（这里一定有蹊跷，对吧？）CSS3 表格行为在 IE7 及更低版本中并没有得到支持，而且也没有稳妥的补救措施。如果你（或者你的客户）愿意摒弃 IE7，那么它就是一个既简单又可靠，而且还很彻底的解决方案。如果真是这样，我绝对推荐你采用这个方案，前面所讲的各种方案就当我没说。

关键是，你甚至都不需要用 `div` 外包装来扮演 `table` 和 `tr` 元素，仅仅是把三栏的 `display` 属性设定为 `table-cell` 就可以了。浏览器的排版引擎在碰到没有表行（`tr`）的一组单元格时，会自动为它们添加表行，而在表行没有被 `table` 元素包围时，会自动为表行添加 `table`。因此，你不需要多写任何标记，只要把每一栏的 `display` 属性设定为 `table-cell`，剩下的事儿就可全部交给浏览器负责了。

因此，要实现一个三栏–流动中栏布局，只需要以下标记：

```
<nav><!-- 内容 --></nav>
<article><!-- 内容 --></article>
<aside><!-- 内容 --></aside>
```

和以下 CSS：

```
nav {display:table-cell; width:150px; padding:10px;
     background:#dcd9c0;}
```

```
article {display:table-cell; padding:10px 20px;
     background:#ffed53;}
aside {display:table-cell; width:210px; padding:10px;
     background:#3f7ccf;}
```

在图 5-5 所示固定宽度布局的 HTML 基础上，我们去掉了内部 div。而在 CSS 中，把每一栏的 float:left 替换成了 display:table-cell，同时去掉了中栏的宽度设定。好了，一个中栏流动、各栏同高，而且能够方便为内容添加内边距和边框的布局就这么出炉了（参见图 5-12）！

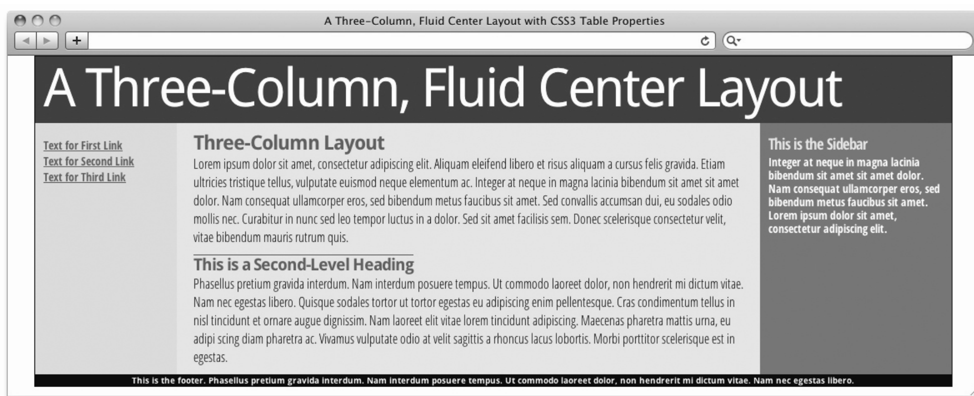


图 5-12　这个流动布局使用了 CSS3 的 display:table-cell，让每个栏形同单元格一般

请注意，这个简单、功能完备的布局对 IE7 和 IE6 可没有任何腻子脚本，甚至连个退化的后备方案都没有。在这些浏览器中，三栏会上下堆叠在一起。因此，除非你下定决心不再支持老版本的 IE，否则就得使用本章前面讲过的其他布局技术。等吧，等到这些浏览器没人用为止。

最后我们总结一下本节讨论的布局技术。这一节先讲了使用内部 div 的浮动固定宽度布局技术，它是适合新旧浏览器的一种最安全的技术。当然，这种技术要求的工作量也最多。而使用 boxsizing:border-box 声明（再加上适用于 IE7 和 IE6 的腻子脚本）则能提供更直观的盒模型，而且不用使用内部 div。最后，CSS3 的 display:table-cell 方案容易实现又功能完善（想流动就流动，想固定就固定，各栏同高，而且不需要内部 div）。然而，它只适合 IE7 以上版本的浏览器。

到目前为止，我们例子中的布局都是在页眉和页脚之间简单地放着三个栏。本章最后，我们在前面三栏布局的基础上，再介绍一个稍微复杂一些的例子。

5.4　多行多栏布局

本章最后，为了给第 7 章设计完整的网页做好准备，我们要把前面学习的三栏布局的技术付诸实用。我准备向大家展示一个更复杂，也更接近实际的页面框架。这个页面的布局可以用作 WordPress 模板或一个公司网站的基础。

前面的布局中只包含一个 `article`、一个 `nav`……，因此比较容易选择，只要用标签名即可。而在创建复杂布局时，一个页面中会出现多个相同的标签，选择的时候就要用上下文选择符来区分它们了。我想通过下面的例子，告诉你怎么给标记中添加最少的 ID 和类，同时精确地选择任意元素。

本章前面说过，使用 CSS3 的 `box-sizing:border-box` 声明，可以避免利用内部 `div` 为内容添加边距。可是，这些内部元素，无论是 `div` 还是别的什么元素，也能辅助添加各种样式，这个例子会展示它们在这方面的用途。

这个布局（如图 5-13 所示）由六个等宽的行组成，其中第四行有三栏，第五行有四栏。

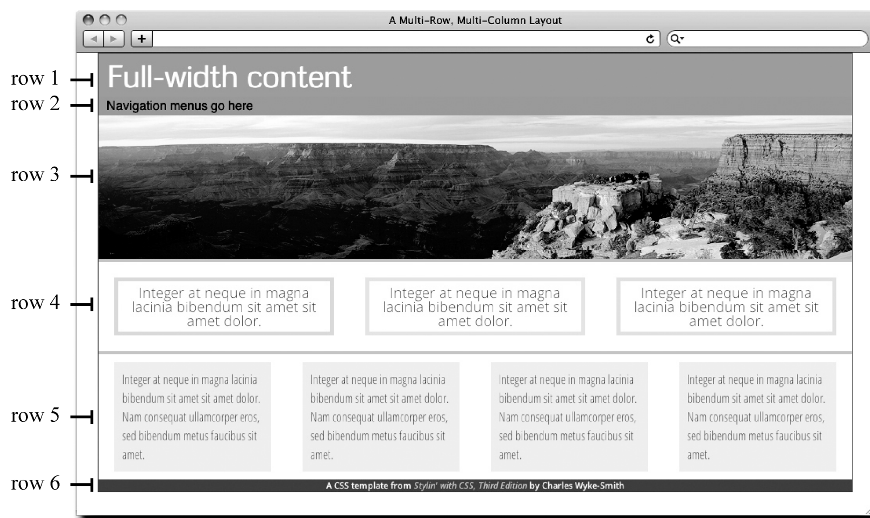


图 5-13　包含六个等宽行和两组不同分栏的布局

这个布局的标记如下。

```
<div id="wrapper">
    <header>
```

```
        <h1>Full-width content</h1>
    </header>
    <nav>
        <p>Navigation menus go here</p>
    </nav>
    <section id="branding">
        <img src="images/grand_canyon.jpg" alt="Grand Canyon" />
    </section><!-- branding 结束 -->
    <section id="feature_area">
        <article>
            <div class="inner">
                <p>Lorem Ipsum text</p>
            </div>
        </article>
        <!-- 省略另外两个 article 元素 -->
    </section><!-- feature_area 结束-->
    <section id="promo_area">
        <article>
            <div class="inner">
                <p>Lorem Ipsum text</p>
            </div>
        </article>
        <!-- 省略另外三个 article 元素 -->
    </section><!-- promo_area 结束-->
    <footer>
        <p>A CSS template from <a href="http://www.stylinwithcss.com"><em>
            Stylin' with CSS, Third Edition</em></a> by Charles Wyke-Smith</p>
    </footer>
</div>
```

至于怎么把以上标记变成图 5-13 所示的布局，实际上我们都已经讲过了。比如，等宽的各行，不用设定它们的宽度，让它们自动扩展填充外包装元素即可。各个栏是由浮动元素构成的，为了防止布局变宽导致右边的栏“下滑”，不要给容器添加内边距，而是把水平内边距加到内部 div 上。是不是？这些你都知道了。与其再把那些 CSS 罗列一遍，不如专门讲几个特殊的 CSS 片段，看看要在包含上百个 HTML 元素的复杂标记中如何精确地选择元素。



本例完整的 CSS 代码，可以在本书网站下载：http://www.stylinwithcss.com。

5.4.1 CSS选择符的实际应用



有必要的话，请读者自行参考第 2 章有关选择符的内容。

随着页面变得越来越复杂，相同的 HTML 元素（如 section、article、nav，等等）会出现很多次——比如，前面布局中的 article 就出现了 7 次。为了选择某个元素，必须区分这些相同的标签名。为此，有些新手会给每个标签都添加一个不同的类名。但这种做法是不值得提倡的。不仅因为类本身就不该这么用（类应该用于标记具有相同特征的元素），而且这么多类会把标记弄得很乱，让 CSS 也很难看懂。为了知道每个类代表哪个元素，你必须不断地查看 HTML。

更好的做法是给标记中每个主要区域的顶级元素添加一个 ID，这也是使用 ID 的正确方式，ID 就是标识页面中唯一元素用的。然后，这些 ID 就会成为 HTML 标记中的“路标”，放在上下文选择符开头的时候，它们就能起到框定后代元素的作用。这就是在标记中保持类和 ID 属性最少的秘诀。而且，相应的上下文选择符也能清晰地传达出路径信息，让人从 CSS 中一眼就能看出它要选择哪个元素。



英文中经常用 hook（这里译为“路标”，也有译为“钩子”的），表示代码中一个唯一的参照点，其他代码通过这个参照点可以与相应的代码交互。

再看看前面的 HTML 标记。其中的三个 ID 是我精确选择任意元素唯一必要的几个“路标”，即使再给布局中添加一些内容元素，恐怕也用不着更多了，顶多再多几个。至于具体怎么使用它们，下面我就通过为第四行的栏添加边框来演示。

这一行的顶级元素是一个 ID 为 feature_area 的 section。这个容器中包含三个 article 元素，分别作为一栏。下面来看看这一行的 CSS。

```
/*每个 article 作为一个浮动栏*/
section#feature_area article {
    float:left;
    width:320px;
    /*对于作为栏的容器，只能添加垂直内边距*/
    padding:10px 0;
    background:#fff;
    border-top:4px solid #f7be84;
}
```

```
/*为所有内容盒子添加公共样式*/
section#feature_area article .inner {
    margin:10px 20px;
    padding:5px;
    background:#fff;
    border:5px solid;
}
/*以下三条分别为三个内容盒子设定样式*/
section#feature_area article:nth-child(1) .inner {
    border-color:#d7dd6f;
}
section#feature_area article:nth-child(2) .inner {
    border-color:#f6dec5;
}
section#feature_area article:nth-child(3) .inner {
    border-color:#d1d8e4;
}
```

我用 section#feature_area article 选择了三个盒子，用一条规则声明了这些元素共有的样式，包括宽度、内边距、边框等。这样，对这三个盒子只需维护一组声明即可。

然后，通过三条规则分别为三个盒子设定独特的样式。下面就是为第二个内部 div 应用样式的规则：

```
section#feature_area article:nth-child(2) .inner {
    border:5px solid #f6dec5;}
```

从后往前，这条规则的意思是：选择类为 inner 的元素，它必须是父元素中的第二个 article，而且这个 article 必须包含在 ID 为 feature_area 的 section 元素中，将它的边框设定为淡橙色。

显然，只要顶级元素上有一个 ID，我们就可以把它作为“路标”，进而选择它的任意后代元素（甚至能够选择将来才会加入其中的内容元素）。另外，同样重要的是，这样也避免了无意中把样式添加给标记中的其他元素上。如果在没有别的手段让你选择某个特定元素的情况下，只在标记中添加一个类或一个 ID，那么 HTML 就能保持清晰整洁，而 CSS 也将易读易维护。

5.4.2 内部DIV实战

这个布局也非常恰当地演示了内部 div 承担的定位和样式这两个角色。具体来说，这个布局中的内部 div 不仅确保了水平内边距不会破坏布局，另外也承担了一个视觉功能——内容区的彩色边框就是它们的边框。它们的父元素 article 在页面中是看不到的，但为了让你看见，我临时给中间的 article 应用一个背景色，如图 5-14 所示。

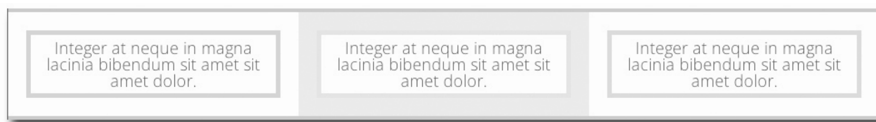


图 5-14 中间的 article 元素拥有灰色背景

从中间带灰色背景的 article 元素可以看出，这一行中的三个 article 充满了容器元素 section，而且彼此之间亲密无间。

再强调一次吧，每一栏中的间距要依靠内部 div。这个例子中的水平间距是由内部 div 的左、右外边距生成的，它们把这个 div 压缩了一下，这才使内容远离了父元素 article（如果此时直接给 article 元素应用水平内边距的话，一定会破坏布局）。而每一栏中的垂直间距是由父元素的内边距生成的。为什么要用父元素呢？原因前面也提到过，就是在父元素没有上、下边框的情况下，子元素的上、下外边距会折叠的。

到此为止，我们关于这个布局的讨论就告以段落了。相信读者对如何使用上下文选择符一定有了深刻的理解。等到第 7 章，我们还将以这个布局为基础，创造出不一样的设计。

5.5 小结

本章，我们知道了可以用浮动的有宽度的元素，以及利用表格行为来创建多栏布局。我们学习了如何让固定布局在页面上居中，也学习了让它们在一定范围内可以伸缩。更重要的是，你知道了如何使用内部 div 在浮动元素中生成间距，而又不会改变布局的总宽度。另外，还掌握了所谓的“人造栏”技术。

你明白了 ID 和类的真正用途，就是不要把它们当成元素的标签，而是要把它们作为上下文中的“路标”。这样，才能让标记中的 ID 和类保持最少。等到第 7 章，我们会在本章最后这个布局的基础上，融合下一章要讨论的界面组件，创造出一个完整的页面框架。第 6 章要介绍的界面组件包括菜单、表单、弹出框，等等。让我们开始吧。

6 界面组件

界面组件是我对 HTML 提供的常见用户界面（UI，User Interface）元素的称呼，包括列表和表单等。实际上，下拉菜单和弹出层也是很常用的界面组件，只不过 HTML 没有提供对应的原生标签而已，本章也将介绍如何通过标记和样式实现它们。

除了介绍怎么实现这些界面组件，我还会告诉你怎么写代码才能更方便地重用这些组件，省得每次都从头写。另外，我还会告诉你怎么把它们放到重用样式表中，以备将来在不同的项目中派上用场。

每个站点都需要导航。那就让我们从导航菜单开始讲吧，看看怎么把 HTML 中的列表变成动态的菜单。



访问本书网站 http://www.stylinwithcss.com 可以在线阅读没有包含在本书中的一章，其中讲到了表格和列表的基础知识，还提供了完整的源代码。

6.1 导航菜单

菜单由一组链接组成。用 HTML 中的列表元素（`ul` 或 `ol`）来分组链接不仅符合逻辑，而且即使没有额外的 CSS 也能适当显示链接的层次。默认情况下，由于列表项（`li`）是块级元素，因此它们会上下堆叠。

6.1.1　纵向菜单

先从最简单的情况开始讲，通过下面这个简单的例子，你会知道把列表转换成菜单的关键技术。

```
<nav class="list1">
    <ul>
        <li><a href="#">Alternative</a></li>
        <li><a href="#">Country</a></li>
        <li><a href="#">Jazz</a></li>
        <li><a href="#">Rock</a></li>
    </ul>
</nav>
```

以上标记给每个列表项的文本都加了链接标签，让它们都变得可以点击。（每个链接中的#是 URL 占位符，将来可以替换成真正的 URL。）

首先，我们通过第一个例子来展示几个问题，很多初学者在为导航链接写样式的时候都会遇到这些问题。然后，再告诉你怎么解决它们。无论如何，这些 CSS 应用到 HTML 之后，会得到一个看起来很简洁的菜单，如图 6-1 所示。

```
/*去掉默认的内边距和外边距*/
* {margin:0; padding:0;}
/*设定本例中菜单的大小和位置*/
nav {margin:50px; width:150px;}
/*给菜单加上边框*/
.list1 ul {border:1px solid #f00; border-radius:3px;
    padding:5px 10px 3px;}
/*去掉项目符号并为链接添加间距*/
.list1 li {list-style-type:none; padding:3px 10px;}
/*“非首位子元素”选择符*/
.list1 li + li {border-top:1px solid #f00;}
/*为链接添加样式*/
.list1 a {text-decoration:none; font:20px Exo, helvetica,
        arial, sans-serif; font-weight:400; color:#000;
        background:#ffed53;}
/*悬停高亮*/
.list1 a:hover {color:#069;}
```

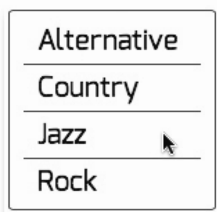


图 6-1　应用样式之后的无序列表，从箭头光标可知，没有文本的区域是不能点击的

添加这些样式没什么难度。HTML5 新增的 nav 元素在语义上很恰当，因此我们用它作为导航菜单的容器，还给它添加了一个类作为设定样式的上下文。

使用“非首位子元素”选择符

前面的 CSS 代码中有两点特别值得一说。首先，加粗的 li + li 选择符的意思是“任何跟在 li 之后的 li”。

对于连续的元素——比如这里多个 li，这个选择符会选择除第一个之外的所有 li 元素。这样，就可以给除第一个 li 之外的所有列表项上方加一条边框。假如我只简单使用 li 选择符，那第一个列表项上方也将被加上边框，而这不是我们想要的。我把这种选择符称为“非首位子元素”选择符，它在选择列表时非常实用。当然，也有实现相同效果的其他方法，比如：

```
/*给所有 li 上方添加一条边框*/
li {border-top:1px solid #f00;}
/*去掉第一个 li 上方的边框*/
li:first-child {border-top:none;}
```

让列表行可以点击

在图 6-2 中，我为每个链接都加上了背景色，以便显示相应链接可点击区域的大小。如图所示，目前只有文本是可以点的，因为链接（a）是行内元素，它会收缩并包住其中的文本。然而，更好的用户体验是让列表项所在的整行都能点击。之所以要重点讲这个问题，是因为我发现很多站点都没有这么做。结果就是用户必须用鼠标点击文本，点击文本之外的地方没有反应。

此外，在每个 li 元素的上方和下方，各有 3 像素的内边距，导致链接文本与列表项边框之间有了距离。正因为有了这段距离，当用户鼠标移动到链接间的间隙时，光标会从“悬停于链接之上”时的小手形状，变成常规的箭头形状。

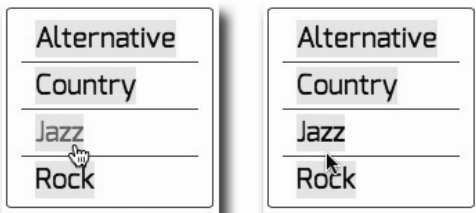


图 6-2　临时加的背景色让我们看清了 a 元素实际可以点击的区域大小，如图所示，光标在经过两个链接的间隙时从小手变成了指针

要解决上述问题，首先得把内边距从 li 元素转移到链接内部，然后让链接完全填满整个列表项。

```
* {margin:0; padding:0;}
nav {margin:50px; width:150px;}
.list1 ul {border:1px solid #f00; border-radius:3px;
padding:5px 10px 3px;}
.list1 li {list-style-type:none; padding:3px 10px;}
.list1 li + li a {border-top:1px solid #f00;}
.list1 a {display:block; padding:3px 10px; textdecoration:
          none; font:20px Exo, helvetica, arial, sansserif;
font-weight:400; color:#000; background:#ffed53;}
.list1 a:hover {color:#069;}
```

把选择符 **li + li** 变成 **li + li a**，就可以把上边框添加到列表项后面的列表项所包含的链接元素上（虽然有点绕，但你应该明白我的意思吧？）。而且，我还把内边距从列表项转移到了链接内部。这样，链接上下就互相接触了，中间没了间隙，鼠标经过时光标状态也不会变了。最后，还要把每个链接的 display 属性设定为 block，这样链接元素的盒子就会由“收缩包围内容”变成“扩展填充父元素”。换句话说，链接的可点击区域将扩大到整个列表行。图 6-3 清晰地说明了经过这些改进后的效果。

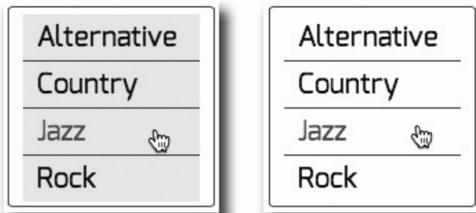


图 6-3　每个链接都填满了整个列表项，所以整个区域都可以点击了。左图的背景色显示了可点击区域，而右图则显示了去掉测试用的背景色之后的最终效果

6.1.2 横向菜单

默认情况下，列表项是垂直堆叠在一起的。不过，要把它们变成水平排列的横向菜单也十分容易。方法就是浮动列表项。比如下面这个简单的列表：

```
<nav class="list1">
    <ul>
        <li><a href="#">Shirts</a></li>
        <li><a href="#">Pants</a></li>
        <li><a href="#">Dresses</a></li>
        <li><a href="#">Shoes</a></li>
        <li><a href="#">Accessories</a></li>
    </ul>
</nav>
```

和下面这些与前面创建纵向菜单很相似的 CSS：

```
.list1 ul {
    /*强制 ul 包围浮动的 li 元素*/
    overflow:hidden;
}
.list1 li {
    /*让 li 元素水平排列*/
    float:left;
    /*去掉项目符号*/
    list-style-type:none;
}
.list1 a {
    /*让链接填满 li 元素*/
    display:block;
    padding:0 16px;
    /*去掉链接的下划线*/
    text-decoration:none;
    color:#999;
}
.list1 li + li a {border-left:1px solid #aaa;}
.list1 a:hover {color:#555;}
```



图 6-4　简单的水平菜单

图 6-4 这个水平菜单是不是很常见？没错，很多时尚的零售网店都有它们的身影。请容我啰嗦几句，解释解释上面的 CSS 代码：浮动让 `li` 元素从垂直变成水平，`display:block` 让链接从收缩变成扩张，从而整个 `li` 元素都变成了可以点击的。另外，选择符 `li + li a` 为除第一个链接之外的每个链接左侧都加了一条竖线，作为视觉分隔线。好啦，可以讲更复杂的样式了。

6.1.3　下拉菜单

在我收到的邮件中，询问下拉菜单问题的最多。所以，本节我就来给大家彻底讲清楚，看一看到底什么是下拉菜单，怎么创建下拉菜单，下拉菜单有什么变化形式。下拉菜单是以一组嵌套列表为基础，综合运用我们刚刚学到的纵向和横向菜单的 CSS 技术创建的。图 6-5 展示了一个下拉菜单。



图 6-5　三级下拉菜单

为方便设定菜单样式，我为列表容器 `nav` 添加了 `multi_drop_menu` 类。这个菜单的每一条 CSS 规则都以 `.multi_drop_menu` 开头，以确保它们只会应用给带有这个类的容器。

下拉菜单的标记是在前面菜单例子基础上扩充而来的。下面就是所有标记，虽然看起来挺吓人，但实际上不过是一个简单的三层嵌套列表而已。

```
<nav class="multi_drop_menu">
    <!-- 一级开始 -->
    <ul>
        <li><a href="#">Power</a></li>
        <li><a href="#">Money</a></li>
        <li><a href="#">Love</a></li>
        <li><a href="#">Fame</a>
            <!-- 二级开始 -->
            <ul>
                <li><a href="#">Sports Star</a></li>
                <li><a href="#">Movie Star</a></li>
                <li><a href="#">Rock Star</a>
                    <!-- 三级开始 -->
                    <ul>
                        <li><a href="#">Bruce Springsteen</a></li>
                        <li><a href="#">Bono</a></li>
                        <li><a href="#">Mick Jagger</a></li>
                        <li><a href="#">Bob Dylan</a></li>
                    </ul>
                    <!-- 三级结束 -->
                </li>
                <li><a href="#">Web Designer</a></li>
            </ul>
            <!-- 二级结束 -->
        </li>
    </ul>
    <!-- 一级结束 -->
</nav>
```

以上标记只包含图 6-5 中展示的菜单项。想嵌套更多子菜单也非常简单，方法是在标记中找到相应链接，把包含子菜单项的无序列表加在链接之后、包含它的 `li` 元素结束标签之前。一会儿你就知道了，这样做无须更改 CSS。下面就从创建水平的顶级菜单开始吧。

顶级菜单

以下是顶级菜单的 CSS（请配合注释理解它们的作用）：

```
/*添加视觉样式*/
.multi_drop_menu {font:1em helvetica, arial, sans-serif;}
.multi_drop_menu a {
```

```
        /*让链接充满列表项*/
        display:block;
        /*文本颜色*/
        color:#555;
        /*背景颜色*/
        background-color:#eee;
        /*链接的内边距*/
        padding:.2em 1em;
        /*分隔线宽度*/
        border-width:3px;
        /*可以有颜色，也可以透明*/
        border-color:transparent;
    }
    .multi_drop_menu a:hover {
        /*悬停时文本颜色*/
        color:#fff;
        /*悬停时背景色*/
        background-color:#aaa;
    }
    .multi_drop_menu a:active {
        /*点击时背景变色*/
        background:#fff;
        /*点击时文本变色*/
        color:#ccc;
    }

    /*添加功能样式*/
    .multi_drop_menu * {margin:0; padding:0;}
    /*强制 ul 包围 li*/
    .multi_drop_menu ul {float:left;}
    .multi_drop_menu li {
        /*水平排列菜单项*/
        float:left;
        /*去掉默认的项目符号*/
        list-style-type:none;
        /*为子菜单提供定位上下文*/
        position:relative;
    }
    .multi_drop_menu li a {
        /*让链接填充列表项*/
        display:block;
```

```
    /*给每个链接添加一个右边框*/
    border-right-style:solid;
    /*背景只出现在内边距区域后面*/
    background-clip:padding-box;
    /*去掉链接的下划线*/
    text-decoration:none;
}
.multi_drop_menu li:last-child a {border-right-style:none;}
/*临时隐藏低级菜单*/
.multi_drop_menu li ul {display:none;}
```



图 6-6　顶级菜单项，此刻鼠标正按住未释放，因此有:active 效果

对于图 6-6 的 CSS 样式，首先要注意的是，我把菜单的视觉样式与功能样式分开来写了。视觉样式控制字体大小、边框和文本的颜色，功能样式控制菜单的布局和行为。两者的区别通过代码注释是很容易分辨的。对于下拉菜单这么复杂的组件，分开来写视觉和功能代码是非常值得提倡的。这样，将来如果有人要修改菜单外观，只要修改它的视觉样式就好了，而功能样式可以原封不动。



如果你在代码中像这样把视觉样式与功能样式分开来写，那一定要用注释说明为什么。

目前，最明显的变化还是 li 通过浮动由垂直堆叠变成了水平排列。至于为了让 ul 包围列表项，没有使用 overflow:hidden，而是使用了 float:left，是因为前者会导致后来添加到下拉菜单中的子菜单无法显示——它们最终会显示在父元素 ul 的外面，结果会因为“溢出”（overflow）而被隐藏（hidden）。

为了保证用户体验，所有视觉样式——内边距、背景、边框，等等，都要应用给链接 a，而不要应用给 ul 或 li，以便热区（可点击区域）最大化，让用户鼠标经过时不会产生前面例子中看到的状态切换。为达到这个目的，同时还要从视觉上分隔链接，我们使用了 background-clip:padding-box 声明，这样就可以阻止链接的背景（像常规状态下一样）延伸到边框后面。然后，让边框透明（也可显示为其他实色），在链接之间产生间隙，让后面的页面能够透过边框被看到。如此一来，不用外边距也能分隔链接，而且鼠标从一个菜单项移动到另一个菜单项时，也不会出现光标切换。

菜单项之间从视觉上是分开，但实际上却是紧挨在一块的。



要了解 background-clip 和透明边框的更多用法，可以阅读这篇文章：http://css-tricks.com/transparent-borders-with-background-clip。

或许你注意到了，我给 li 元素应用了 position:relative，这是给添加子菜单作准备的，在当前这一步里实际上它没有什么效果。另外，最后一行 CSS 隐藏了子菜单，以防显示它们影响我们创建顶级菜单。在后面两步中，我会告诉你怎么在菜单项处于悬停状态时显示它们。

第一步为顶级菜单添加的样式都会被次级菜单继承。换句话说，需要应用给子菜单的样式大部分都已经写完啦！

菜单的下拉部分

虽然子菜单继承顶级菜单的样式为我们提供了方便，但其实有些样式在子菜单中并不是必要的。比如，（列表的第二级）下拉菜单是垂直堆叠的，所以我们不希望其中的列表项继承浮动而变成水平排列。另外，顶级菜单使用右边框作为分隔线，而下拉菜单需要使用上边框来分隔垂直堆叠的菜单项。这一步，我们还会绝对定位下拉菜单，以便它能精确地对齐父元素（包含子菜单的 li），而这个父元素已经在上一步被设定成相对定位了。

与此同时，为了看到这些设定的效果，还要显示二级下拉菜单，并继续隐藏三级下拉菜单（下一步再考虑它）。

以下是实现上述改变需要添加的 CSS：

```
/* 添加的视觉样式 */
/*二级菜单宽度*/
.multi_drop_menu li ul {width:9em;}
.multi_drop_menu li li a {
    /*去掉继承的右边框*/
    border-right-style:none;
    /*添加上边框*/
    border-top-style:solid;
}
/* 添加的功能样式 */
.multi_drop_menu li ul {
```

```
    /*临时显示二级下拉菜单*/
    display:block;
    /*相对于父菜单项定位*/
    position:absolute;
    /*左边与父菜单项对齐*/
    left:0;
    /*顶边与父菜单项底边对齐*/
    top:100%;
}

.multi_drop_menu li li {
    /*停止浮动，恢复堆叠*/
    float:none;
}
.multi_drop_menu li li ul {
    /*继续隐藏三级下拉菜单*/
    display:none;
}
```

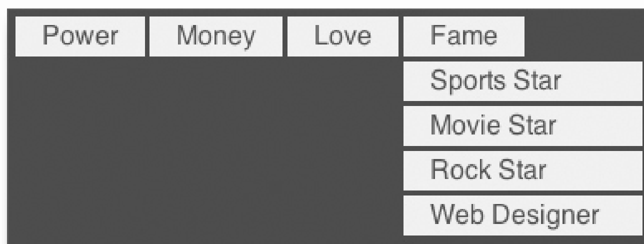


图 6-7　二级下拉菜单已经显示在其父菜单项下方

这一步成功的关键是下拉菜单的绝对/相对定位。通过将其顶边位置（top）设定为 100%（相对于其相对定位的父元素 li），其顶边会与父元素底边恰好对齐。它与父元素之间的间隙，实际上是下拉菜单中第一个链接的边框，参见图 6-7。

让下拉菜单响应鼠标事件

接下来是好玩的部分了，即要让下拉菜单拥有自己的功能。换句话说，它一开始是隐藏的，只有在其父元素处于鼠标悬停状态时，才需要显示。

```
.multi_drop_menu li ul {
    /*隐藏二级下拉菜单*/
    display:none;
```

```
    /*相对于父菜单项定位*/
    position:absolute;
    /*左边与父菜单项对齐*/
    left:0;
    /*顶边与父菜单项底边对齐*/
    top:100%;
}
.multi_drop_menu li:hover > ul {
    /*父元素悬停时显示*/
    display:block;
}
```



图 6-8　鼠标移动到链接上，其子菜单就会显示出来

让菜单起作用的关键在于先把它藏起来：

```
/*隐藏二级菜单*/
li ul {display:none;}
```

然后，再在父元素悬停时把它显示出来：

```
/*显示二级菜单*/
li:hover > ul {display:block;}
```

最后一行 CSS 的意思是：当鼠标移动到列表项上时，就显示它的子列表。注意，这里的:hover 触发器是设定在 li 元素而非链接自身上的。这样做是因为我们想要显示 li 的子元素 ul，而它不是想链接的子元素。此外，为了只显示其子元素，悬停列表项与子列表之间还有一个子选择符>，结果如图 6-8 所示。如果没有这个子选择符，当顶级菜单项处于悬停状态时，会同时显示二级和三级菜单。

添加三级菜单

接下来我们就趁热打铁，把三级菜单也弄好吧。实际上，这时候三级菜单已经算是能用了。不信？找到图 6-8 的示例代码，在浏览器中打开，然后用鼠标触发子菜单，就会看到图 6-9 所示的效果。

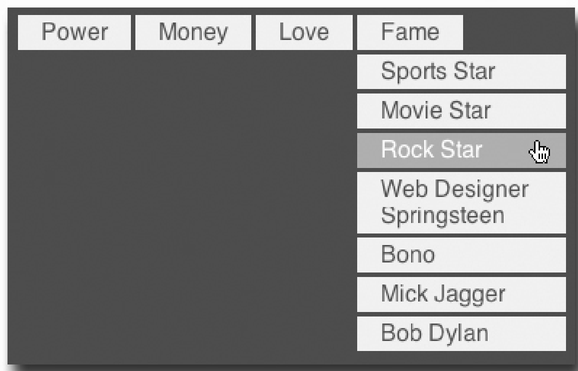


图 6-9　在父元素处于鼠标之下时，三级菜单也会显示，只不过位置不对

由于前面包含:hover 的 CSS 规则会像应用给二级菜单一样，应用给三级菜单，所以在父元素处于鼠标之下时，三级菜单自然也会显示出来。可是，图 6-9 中的三级菜单被其父元素挡住了。这时候三级菜单与其二级父元素的位置是什么关系？没错，就是二级菜单与顶级菜单的关系。所以，三级菜单跑到了鼠标下的二级菜单后面去了，而且其第一项顶边与悬停的父菜单项底边是对齐的。从图中可以看到，三级菜单第一项“Bruce Springsteen”的上半部分被二级菜单最后一项“Web Designer”给盖住了。

我们在这一步要做的，就是把三级菜单放到二级菜单右侧，让它的顶边与鼠标所在菜单项的底边对齐。

```
.multi_drop_menu li li ul {
    /*相对于父菜单定位*/
    position:absolute;
    /*与父菜单右侧对齐*/
    left:100%;
    /*与父菜单项顶边对齐*/
    top:0;
}
```

现在的菜单能用了，参见图 6-10。你可以在顶级添加更多列表项，在标记中添加更多子列表，然后无须多写一行 CSS，它们就能构成新的下拉菜单。此时，我还想再作两个调整。首先，为了真正让这些代码有用，而且可以重用，需要再写一些样式，让顶级菜单能够垂直显示，以便能将其用在导航侧边栏里。为此，我得先给 nav 容器添加第二个类 vertical。

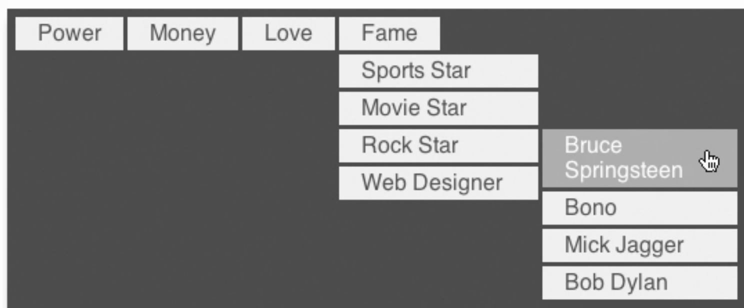


图 6-10　三级下拉菜单显示在了二级菜单旁边

```
<!-- HTML 类名之间要有空格 -->
<nav class="multi_drop_menu vertical">
```

这样，我就可以写一些样式，让它们只在导航容器有 vertical 类的情况下才起作用。所以，这些新增规则的选择符开头都是

```
/*CSS 类名之间没有空格*/
.multi_drop_menu.vertical
```

这意味着选择一个带有两个类的元素——注意，CSS 的类名间没有空格。

```
/*顶级垂直菜单宽度*/
.multi_drop_menu.vertical {width:8em;}
.multi_drop_menu.vertical li a {
    border-right-style:none;
    border-top-style:solid;
}
.multi_drop_menu.vertical li li a {border-left-style:solid;}
.multi_drop_menu.vertical ul,
.multi_drop_menu.vertical li {
    /*让顶级菜单垂直显示*/
    float:none;
}
.multi_drop_menu.vertical li ul {
    /*子菜单左边与上一级菜单右边对齐*/
    left:100%;
    /*子菜单顶边与上一级菜单项顶边对齐*/
    top:0;
}
```

为了让菜单恢复默认的堆叠状态，这里重置了顶级 li 元素及其父元素 ul 的浮动属性，后者原来浮动是为了包围浮动的 li 元素。

这里还为 nav 容器设定了宽度。如果不设定导航容器宽度，那么 nav 及其包含的顶级菜单项都会尽可能伸展。最后，我们把二级菜单与顶级菜单的位置关系，设定成了前面三级菜单和二级菜单之间的关系，即让子菜单顶边与父菜单项顶边对齐。当然，还去掉了顶级菜单项的右边框，代之以上边框，如图 6-11 所示。

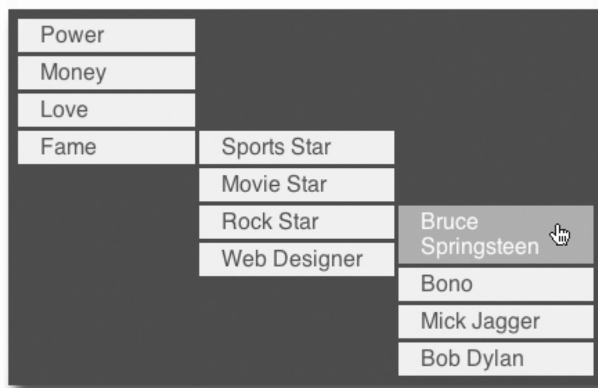


图 6-11 增加了垂直菜单样式后，顶级菜单项上下堆叠起来了

突出显示选择路径

图 6-11 显示，只有位于鼠标下方的元素才会突出显示。为了让用户明确地知道自己是怎么一路选择下来的，还需要让每一级菜单中被选择的元素突出显示。实际上，很简单，只要把.multi_drop_menu a:hover 替换成以下 CSS 即可。

```
.multi_drop_menu li:hover > a {
    /*悬停时的文本颜色*/
    color:#fff;
    /*悬停时的背景颜色*/
    background-color:#aaa
}
```

这个小小的替换能够起作用，是因为:hover 事件会沿着元素的结构层次“向上冒泡”。所以，把:hover 设定在 li 元素，就相当于也把它设定给了所有祖先 li 元素。然后，只要再给其子元素（链接）设定样式即可。这个改进能够极大地提高菜单的易用性，而我们的下拉菜单到此也讲完了，最终结果如图 6-12 所示。

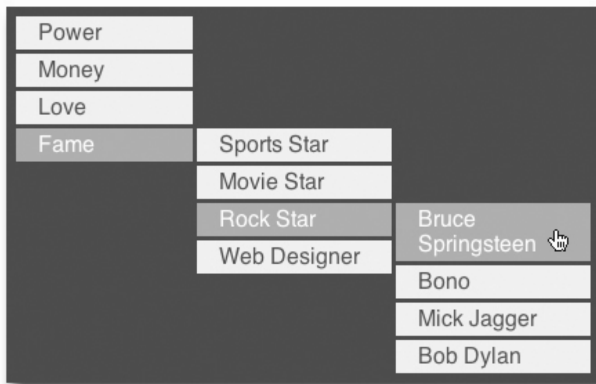


图 6-12　每一级菜单中的被选项都突出显示出来了

有了以上 CSS 文件，只要把它链接到页面中，并给一个包含无序列表的容器加上 `multi_drop_menu` 类，该列表就会摇身一变，成为一个全功能的菜单，这个魔术我会在下一章变给你看。

6.2　表单

表单与其他页面元素的作用不同。其他元素是把服务器发过来的内容显示给用户，而表单则是把用户的信息发送给服务器。作为设计者，我们当然希望用户能够顺利地使用表单，从而得到他们的反馈、意见、联系信息……当然——还有他们的信用卡号。大型表单的复杂程度较高，电商网站的购物车就是一例。假如设计得不好，让用户觉得费解，很可能转眼之间就丢掉了生意。可见，设计的品质是可以用白花花的银子来衡量的。



说起设计表单，我强烈向大家推荐 Luke Wroblewski 专门写表单的书 *Web Form Design: Filling in the Blanks*：http://www.lukew.com/resources/web_form_design.asp。

处理表单数据的话题超出了本书的范围，本章的重点在于告诉你怎么把表单设计得简洁易用。好吧，先看一个完整的表单，找找感觉。



想了解怎么在服务器端验证表单数据？可以看看我的另一本书 *Codin' for the Web*，其中讲到了 PHP 编程和 MySQL 数据库。

6.2.1　HTML表单元素

以下是完成后的表单，本节正是要介绍怎么设计这个表单。

HTML5 与表单

HTML5 为 input 元素新增了 13 种新类型（type 属性），也对表单进行了大幅增强，包括特别特别棒的 placeholder 属性，可以在文本框中显示说明文字（例如“在这里填写用户名”），只要用户一开始输入它就会消失。

要了解新增的 HTML5 表单标签和属性，以及浏览器对它们的支持情况，可以参考这个网页 http://wufoo.com/html5，还有这个 http://www.html5rocks.com/en/tutorials/forms/html5forms。

我们先把这个表单的所有标记展示一下，然后再讨论其中个别的 HTML 元素。

```
<!-- 必要的 form 标签 -->
<form class="stylin_form1" action="process_form.php" method="post">
    <h3>A Stylin' Form</h3>
    <!-- 控件组，即各种控件的容器 -->
    <fieldset>
    <!-- 控件组的文字说明，或标题 -->
    <legend><span>Part 1 &#8226; Basic Controls</span></legend>
    <!-- 开始单行文本输入控件 -->
    <section>
        <p class="note">* indicates required field</p>
        <!-- for 属性把标注与控件关联起来，它的值必须与控件 ID 值相同 -->
        <label for="user_name">User Name<span> *</span></label>
        <!-- text 属性让这个控件可以输入文本 -->
        <input type="text" id="user_name" name="user_name" />
        <p>Please select a user name</p>
    </section>
    <!-- 开始密码控件 -->
    <section>
        <label for="password">Password<span> *</span></label>
        <!-- 密码文本显示为掩码 -->
        <input type="password" id="password" name="password" maxlength="20" />
        <p>Password must be 8 or more characters</p>
    </section>
```

```
<!-- 开始多行文本输入控件 -->
<section>
    <label for="description">Description</label>
    <textarea id="description" name="description"
              placeholder="Enter the description here."></textarea>
</section>
<!-- 开始 HTML5 日期控件 -->
<section>
    <label for="description">Date</label>
    <input type="date" id="special_date" name="event_date" min="2012-09-05" />
</section>
</fieldset>
<fieldset>
<legend><span>Part 2 &#8226; Multiple-Choice Controls</span></legend>
<!-- 开始复选框 -->
<section>
    <h4>Select Any Number</h4>
        <section>
            <input type="checkbox" id="check1" name="checkset" value="1"
                tabindex="4" />
            <label for="check1">Choice 1</label>
        </section>
        <section>
            <input type="checkbox" checked="checked" id="check2" name=
                "checkset" value="2" />
            <label for="check2">Choice 2 is pre-checked</label>
        </section>
        <section>
            <input type="checkbox" id="check3" name="checkset" value="3" />
            <label for="check3">Choice 3&mdash;add as many as you need!
                </label>
        </section>
    <p>You must choose one or more</p>
</section>
<!-- 开始单选按钮 -->
<section>
    <h4>Select Only One</h4>
    <section>
        <input checked="checked" id="radio1" name="radioset" type="radio"
            value="Choice_1" />
```

```
            <label for="radio1">Choice 1 is pre-selected and shows the text
                wraps nicely if it goes to multiple lines.</label>
        </section>
        <section>
            <input id="radio2" name="radioset" type="radio" value="Choice_2" />
            <label for="radio2">Choice 2</label>
        </section>
        <section>
            <input id="radio3" name="radioset" type="radio" value="Choice_3" />
            <label for="radio3">Choice 3</label>
        </section>
    </section>
    <!-- 开始选项列表（下拉列表） -->
    <section>
        <label for="select_choice">Select Your Choice</label>
        <select id="select_choice" name="select_choice">
            <option value="0">None</option>
            <option value="1">Choice 1</option>
            <option value="2">Choice 2</option>
            <option value="3">Choice 3</option>
            <option value="4">Choice 4</option>
        </select>
    </section>
    </fieldset>
    <!-- 开始提交按钮 -->
    <section>
        <input type="submit" value="Submit This Form" />
    </section>
</form>
```



要了解关于 input 元素更详细的介绍，可以参考这两篇文章：http://htmlhelp.com/reference/html40/forms/input.html，还有 http://www.javascript-coder.com/html-form/html-form-tutorial-p1.phtml。

图 6-13　基于常用表单元素创建的表单

好啦，马上就来讲一讲这个表单的标记，先从最开头的 form 元素讲起。

form 元素

所有表单的标记都包含在一个 form 元素中。

```
<form class="stylin_form1" action="process_form.php" method="post">
```

```
    <!-- 这里是表单标记 -->
</form>
```

form 元素有两个必要的属性：action 和 method。action 属性用于指定服务器上用来处理表单数据的文件的 URL。method（值要么是 post，要么是 get）用于指定怎么把数据发送到服务器。



想学习服务器如何处理表单数据？推荐看看这个网页：http://www.javascript-coder.com/html-form/html-form-tutorial-phtml。这个网页底部还有不少与表单相关的链接，也值得一读。

提交表单

用户提交表单后，他们在表单控件中填写的表单数据或者做出的选择，都会被发送到服务器。所谓控件，是对表单中用来收集数据的各种表单组件的通称，包括文本框、复选框、单选按钮，等等。表单中的数据是以“名=值”的形式发送给服务器的，比如“username=chrisconsumer”，每个控件都是这么一个名/值对形式。这里的“名”就是你在控件 name 属性中设定的名字。“值”可能是用户在文本控件中输入的信息，也可能是表示可选控件（如复选框）中某一项是否被选中的布尔状态值（比如某个复选框中用 1 表示选中，用 0 表示未选中）。



在确定这些控件的名字时，最好跟编程人员以及数据库管理员共同协商，得到一个大家都认可的命名方案。

控件组

可以把一组相关的表单控件组织到一个控件组元素 fieldset 中。比如，在一个电商网站上，用于收集用户姓名和地址的控件组，可能会加上一个标题叫“收货人信息”。随后可能就是另一个用于填写信用卡信息的控件组。

前面表单标记使用了两个 fieldset 元素，把所有控件分成了两组。第一组包含基本的标注和控件，第二组包含更复杂的控件，比如复选框（type="checkbox"）、单选按钮（type="radio"）和选项列表（select）。fieldset 的第一个子元素一定得是 legend 文本元素，其中包含这个控件组的标题。

```
<fieldset>
    <legend>Mailing Address</legend>
```

```
    <!--- 标注和控件 -->
</fieldset>
```

从图 6-13 中可以看到，我给两个控件组的 legend 元素都加了灰色条，文本则反白显示，从而明确表示它们是两个独立的表单区块。

控件与标注

表单包含一或多个控件。如前所述，控件就是让用户勾选、点选、输入的组件。对于那些可以输入的控件，大家又俗称其为字段。每个表单控件（submit 按钮除外），都有一个对应的 label 文本元素，用于描述控件代表的数据。以下代码和图 6-14 展示了这个规范的结构。

```
<label for="user_name">User Name</label>
<!-- 创建一个文本字段 -->
<input type="text" id="user_name" name="user_name" />
<p>Please select a user name</p>
```



图 6-14　表单控件及其标注：这里是一个文本字段，通常都应该搭配一个标注告诉用户在里面填写什么内容

label 元素可以包含控件，也可以放在控件前头或后头。如果像前面那样没有用 label 包含控件，那么 label 的 for 属性与控件的 id 属性必须匹配，以便把两者关联起来。不过，控件与标注之间的关系是隐式存在的，而且如果 label 像下面这样包含控件，其 for 属性也不是必需的：



for 属性与 id 属性通过相同的值关联起来后，用户点击标注文本也可以选择单选按钮和复选框。

```
<label>User Name<input type="text" name="user_name" />
</label>
```

无论采用哪种方式关联标注与控件都没问题，重要的是一定要有这种关联。这种关

联对于屏幕阅读器等辅助技术解读表单非常关键。我个人习惯于把 label 放在控件前面（当然对于复选框和单选按钮，还是要放在后面），因为这样更方便设定样式。

除了标记和控件，有时候还需要告诉用户如何填写或选择，特别是在需要填写特定格式数据（如日期）的时候。HTML 中没有为表单控件说明规定什么标签，因此我们就用 p 这个段落元素吧。另外，说明的文字最好方便在用户输入无效数据的情况下突出显示（比如变成红色），这样就不用单独准备和控制错误消息了。本例后面会介绍怎么做到这一点。



当然，如果你希望在用户输入无效数据时显示不同于说明的错误消息，可以再添加一个 p 元素，并赋予它.error 类。

控件类型

最常用的 HTML 表单元素是什么？你一定会说是 input。一点都没错儿！这个元素之所以常用，是因为它可以在屏幕上显示为多种不同的外观，并具有不同的行为。文本框、复选框、单选按钮等的背后都是 input 元素，区别在于它们的 type 属性。以下列出了 type 属性部分可能的值。

- text：基本的单行文本框。
- password：文本显示为掩码。
- checkbox：复选框。
- radio：单选按钮。
- submit：提交表单的按钮。
- time、date、search：HTML5 文本框的变体。



要想知道 input 元素的 type 属性的所有值，可以参考这里：https://developer.mozilla.org/en-US/docs/HTML/Element/Input。

实际上，以上几种类型都是前面表单标记中用到的，这些类型值决定了相应 input 元素的外观和行为。

有一个文本控件不是 input 元素的变体，那就是多行文本区 textarea 元素。另外，在用户输入之前一直显示的占位符文本，是通过 placeholder 属性设定的。

复选框、单选按钮和选项列表

这三种表单元素比一般的控件/标注组合要复杂一些。

- 复选框可以让用户从多个备选项中选择一或多个项。每个备选项都是非排他型的，即勾选同一组中的其他项，不会影响已经选择的项。
- 单选按钮限制用户只能从多个备选项中选择一项。每个备选项都是排他型的，即勾选同一组中的另一项，就会取消对前一项的选择。

  要特别注意一下，复选框和单选按钮都是成组出现的，分组的方法是为它们设定相同的 name 属性。对于本例中的单选按钮，它们共同的名字叫 radioset。这个组中的每个单选按钮由其 value 属性唯一标识。在本例中，分别是 Choice_1、Choice_2，等等。假设用户选择了第一个按钮，那么发送到服务器的名/值对，就是 radioset=Choice_1。

- 选项列表（select 元素）会创建一个下拉列表，用户可以在其中点选。在 select 元素中，每个备选项都用 option 文本元素来生成。



要知道 HTML 中的所有表单元素，可以参考这里：http://reference.sitepoint.com/html/elements-form。

6.2.2　表单标记策略

因为 HTML 标准没有规定专门包含控件及其标注的元素，所以我就用块级 section 元素来充当这个角色。这样既方便组织控件，又方便为行内表单控件及其标注设定样式。

基本的标注和控件

在前面的标记中，我把每个标注和控件都放到了一个 section 元素中，让 label 位于控件之前。

```
<section>
    <label>…</label>
    <input />
    <p>…</p> <!-- 控件使用说明 -->
</section>
```

通过把每一对标注和控件都放到块级 section 元素中,它们自然就在页面上垂直堆叠起来了。而且，这样也有了一个可以定位它们的包含元素。

复选框和单选按钮

单选按钮和复选框是由很多标注/控件组成的，所以我把它们都放到了一个 section 元素中（便于为标注/控件设定基本的 CSS 样式）。然后，在这个元素里面，再把每个复选框或单选按钮的标注/控件分别放到各自的 section 元素中。

```
<section>
    <h4>Set Heading</h4>
    <!-- 第一个单选按钮/复选框 -->
    <section>
        <input />
        <label>…</label>
    </section>
    <!-- 第二个单选按钮/复选框 -->
    <section>
        <input />
        <label>…</label>
    </section>
    <!-- 控件使用说明 -->
    <p>…</p>
</section>
```

这时候，我把标注放到了控件*后面*，这样它才能显示在控件右侧。虽然每个控件（复选框和单选按钮）都有一个对应的标注，但不能将它们用作整个控件组的标题。每个 label 元素只能用于标注一个控件。为此，我就用一个 h4 元素来作为它们的标题。如前所述，我还添加了一个 p 元素，用于显示控件使用说明。HTML 未来的版本中或许会专门规定这些特定用途的标签。但在此之前，到底选用什么标签来达到目的，还是会因人而异。好了，该看一看实现最终表单的 CSS 样式了。

6.2.3　设定表单样式

首先，从表单的整体布局开始。换句话说，就是先设定 form 和两个 fieldset 元素的样式。

```
form.stylin_form1 {
    width:14em; /* 表单整体宽度 */
    margin:20px auto; /* 在容器内居中 */
    border:1px solid #bbb7ae;
    padding:.5em .5em .15em;
    }
.stylin_form1 h3 { /* 表单主标题 */
    margin:0;
    padding:0 0 .2em .2em;
    font-weight:600;
    color:#bbb7ae;
    }
.stylin_form1 fieldset { /*包含控件与标注*/
    margin:0;
    padding:0 0 .2em 0;
    width:100%;
    border:0;
    }
/* legend 元素的位置不同寻常，所以我把它的文本包含在一个块级 span 中，转而为 span
设定样式 */
.stylin_form1 legend {
    width:100%;
    padding:.3em 0;
    background:#bbb7ae; /*灰色条*/
    }
.stylin_form1 legend span { /* 设定标题文本的样式 */
    display:block;
    font-size:1em;
    line-height:1.1em;
    padding: 0 0 0 .4em;
    font-weight:700;
    color:#fff; /*灰色条上的反白文本*/
    }
```

这样，表单就变成了如图 6-15 所示。

如图 6-15 所示，通过为 form 设定内边距，它所包含的内容都与表单边界空开了距离。很明显，我们给 legend 添加了灰色条，相应地把文本设定成了白色。legend 元素默认的位置是由浏览器内部的一种未加说明的机制确定的，并不是由浏览器样式表设定的。因此，不可能通过 CSS 来精确控制它的位置。解决这个问题的方法，就是把

它的文本包含在一个 span 元素中，将该元素设定为 display:block，然后再设定该元素的位置。接下来，我们从宏观到微观，再看看怎么给控件组（fieldset）中的控件和标注设定样式。

图 6-15　为 form 和 fieldset 元素设定了样式之后

```
.stylin_form1 section {
    overflow:hidden; /* 强制 section 包含表单控件及标注 */
    padding:.2em 0 .4em 0;
    border-bottom:8px solid #e7e5df; /* 根据需要在每个 section 间增加间距 */
    }
.stylin_form1 section:last-child { /* 每组最后一个 section 没有边框 */
    border-bottom:0px;
    }
.stylin_form1 section label,    /* 表单控件的标注 */
.stylin_form1 section h4 { /* h4 是复选框和单选按钮组的标题 */
    display:block;
    clear:both;
    margin:.3em .3em 0 0; /* 右外边距确保标注文本在碰到 input 之前会换行 */
    padding-bottom:.1em;
    font-size:.8em;
    font-family:'Droid Sans';
```

```
        font-weight:400;
        line-height:1.1;
        }
    .stylin_form1 section label span, /* 星号表示必填字段 */
    .stylin_form1 section h4 span {
        font-size:.75em;
        vertical-align:text-top;
        color:#f00;
        }
    .stylin_form1 section p.note { /*说明星号是必填字段的文本*/
        font-size:.7em;
        color:#f00;
        margin:0;
        padding:0 0 .3em 0;
        }
    .stylin_form1 section input,
    .stylin_form1 section textarea,
    .stylin_form1 section select {
        margin:.2em .5em .2em 0;
        padding:.2em .4em; /* 给 input 中的文本添加间距 */
        color:#000;
        box-shadow:1px 1px 3px #ccc;
        font-size:.8em;
        /* 针对 Firefox - 没有这条声明会在 textarea 上使用 Courier */
        font-family:inherit;
        outline:none; /* 去掉默认蓝色聚焦轮廓线 */
        }
    /* 设定文本字段（文本、密码、日期、文本区等）的样式，并加圆角 */
    .stylin_form1 section input,
    .stylin_form1 section textarea {
        width:12em; /*设定字段宽度*/
        border:1px solid #bbb7ae;
        border-radius:3px; /*圆角边框*/
        }
    .stylin_form1 section textarea {
        height:5em;  /* textarea 的高度 */
        margin-top:.3em; /* 与上面 label 的间距 */
        line-height:1.1;
        }
    .stylin_form1 section p { /*控件使用说明*/
        margin:.3em .75em 0;
```

```
    clear:both;
    font-size:.7em;
    line-height:1.1;
    color:#000;
    }
.stylin_form1 section p.error {
    color:#f00;  /* 添加 error 类，把说明文字设定为红色 */
    }
```

图 6-16　第一个 fieldset 中的控件和标注都被应用了样式

如图 6-16 所示，我把所有控件的可见盒子（类型为 text、date、textarea 和 select）都设定为相同宽度，为说明文字应用了样式，并给每个 section 元素添加了较宽的边框，让它们从视觉上保持适当距离。

某些标注后面的星号表示相应字段是必填的，也就是用户不能不填。这个（用 Shift-8）输入的星号被包含在 span 元素中，以便调整它与标注文本的相对位置，并应用不同颜色。注意，我们还为控件使用说明段落设定了一条 p.error 规则。如果控件中数据无效，这个类就会被添加到说明段落上。然后，说明文字会变成红色，提醒用户必须输入有效数据才能提交表单。



这个 error 类将由验证表单的代码添加。

好，下面该轮到复选框、单选按钮和选项列表了。这些控件全都位于第二个控件组中。从图 6-16 底部可以看到，它们的标注都在控件下方，而我想让这些标注与控件并排显示。先别急着看下面的 CSS 规则，建议你此刻翻到 6.2.2 节，重温一下复选框和单选按钮区块（section）的标记结构。

```
.stylin_form1 section section { /* 控件/标注的内包装 */
    overflow:hidden; /* 强制元素包围浮动标注 */
    margin:.2em 0 .3em .4em;
    padding:0 0 .1em 0;
    border-bottom:none;
    }
.stylin_form1 section section input { /*单选按钮或复选框*/
    float:left;
    clear:both;
    width:auto; /* 重设继承的宽度 */
    margin:.1em 0 0em .3em; /* 顶部与标注对齐，左侧防止 intput 溢出 */
    }
.stylin_form1 section section label {
    float:left;
    clear:none; /* 重设继承的值 */
    width:15em;
    margin:.15em 0 0 .6em; /* 在相邻的复选框之间、复选框与标注之间增加间距 */
    font-weight:normal; /*  重设继承的值 */
    font-size:.7em;
    line-height:1.2;
    }
.stylin_form1 section select {
    margin-left:.4em;
    font-size:.85em;
    }
.stylin_form1 section input[type="submit"] { /*提交按钮*/
    width:auto; /* 覆盖为其他字段设定的宽度 */
    margin:.4em .3em 0 0;
    font-size:1em;
    font-weight:800;
    color:#fff;
    background-color:#bbb7ae;
    cursor:pointer; /*在鼠标位于按钮之上时，把光标变成小手形状*/
    }
.stylin_form1 > section:last-child { /*居中提交按钮*/
    text-align:center;
    }
```

每一组单选按钮和复选框的标题，看起来很像表单第一个控件组里的标注。但实际上，它们是 h4 元素——我们只能用 label 来关联控件。正因为如此，要把同样的样式设定给 h4 很简单,把为表单第一个控件组中的 label 设定样式的选择符中的 label 替换成 h4 即可。然后，给这里每个复选框和单选按钮的小标注另外写一种样式。为了让选项列表与其他元素对齐，我们也为它设定了相应的样式，通过减小其文本大小也把它缩小了一点。至于提交按钮，它的宽度不再继承前面设定给其他 input 的规则,并且在表单底部居中。明确告诉用户，这是表单的最后一个步骤，参见图 6-17。



在为 name 属性设定同一个名字的情况下，复选框和单选按钮的行为就像一个组一样。提交表单时，它们的 name 值与 value 值，将分别作为“名=值”对中的“名”和“值”。

Part 2 · Multiple-Choice Controls
Select Any Number
Choice 1
Choice 2 is pre-checked
Choice 3—add as many as you need!
You must choose one or more
Select Only One
Choice 1 is pre-selected and shows the text wraps nicely if it goes to multiple lines.
Choice 2
Choice 3
Select Your Choice
None
Submit This Form

图 6-17　复选框、单选按钮和提交按钮已经各就各位

到现在为止，“标注在上”版的表单已经完成了。接下来，我会再向大家展示一些样式，它们能让表单的标注显示在控件的左侧。使用这个“标注在左”版的样式也很简单，只要给 form 元素加上 labels_left 类即可。

```
form.stylin_form1.labels_left {
    width:22em; /*加宽表单，为标注腾出地方*/
    }
```

```
form.stylin_form1.labels_left label,
form.stylin_form1.labels_left h4 {
    float:left; /*把标注浮动到控件左侧*/
    width:8em;
}
form.stylin_form1.labels_left p {
    margin:0 0 0 9.35em; /*缩进控件说明，以便它位于控件正下方*/
    padding:.3em 0 0 0;
    clear:both; /*确保说明不会跟着浮动的标注和控件走*/
    }
form.stylin_form1.labels_left p.note { /*必填字段文本下方的间距*/
    margin:0 0 .2em 0;
    }
/*每个单选按钮或复选框及其标注的内包装*/
form.stylin_form1.labels_left section section {
    width:10em;
    margin-left:6.5em;
    padding-top:0;
    }
form.stylin_form1.labels_left section section input {
    width:1.25em; /*单选按钮或复选框的宽度*/
    margin-left:0;
    }
.stylin_form1.labels_left section input,
.stylin_form1.labels_left section textarea,
.stylin_form1.labels_left section select {
    float:left; /*让控件成为第二栏*/
    width:12em;
    }
.stylin_form1.labels_left section select { /*缩进选项列表*/
    margin-left:.2em;
    }
/*防止提交表单按钮继承浮动的行为*/
.stylin_form1.labels_left > section input[type=submit] {
    float:none;
    }
```

图 6-18 展示了这两个版本的最终效果。其中，“标注在左”版为控件使用说明元素（p）应用了 error 类。

图 6-18　表单的两个版本，左边标注在上，右边标注在左且突出显示了控件使用说明

通过前面这个例子涉及的那么多 HTML 和 CSS 代码，你一定深深体会到了为表单添加样式多么复杂，多么耗时间。正因为如此，我对这里原型式表单的标记进行了精心设计，而且写了很多注释，以方便你轻松地把它作为你自己设计表单的起点。直接复制粘贴 HTML 代码也好，模仿其结构再添加你需要的各种控件也罢，别忘了把这个表单 CSS 链接到页面中。然后，给你的 `form` 元素加上 `stylin_form1` 类，你的表单一下子就能变成现在这个样子。以此为起点，进行各种改进和增强，就会非常容易了。我自己就是这样做的，这样每次都能节省好几个钟头的时间。下面，我们谈一谈如何设计搜索表单吧。

6.2.4　设计搜索表单

几乎每个站点都会提供一种搜索机制。我猜，恐怕你很难把一个搜索框当成一个表单，但它的确是——一个字段的表单。搜索表单提供了简单的搜索功能，它随时恭候在标题栏的右侧，那儿几乎是它的专属位置。苹果网站上那个小巧、低调的搜索框，在用户点击它的时候会扩展开来，为输入关键词提供更多空间。它旁边甚至连一个按钮都没有——用户都知道点击、输入，然后再按 Return 或回车键。这种搜索表单的 form 元素中只需要包含一个 label 和一个 input。在下面的例子中，我会把这个表单放在一个 header 标签中，以便得到一个包含搜索功能的页眉。

```
<header>
    <form class="stylin_form_search1" action="#" method="post">
        <label for="search">search</label>
        <input type="search" id="search" name="search"
        placeholder="search" />
    </form>
</header>
```

借这个例子，我们正好可以讲一讲 CSS3 的过渡功能。过渡可以基于某个 CSS 属性实现动画效果。在这个例子中，我们要利用它实现文本框动态变宽的效果。对应的 CSS 如下：

```
* {margin:0; padding:0;}
header { /*在这个例子中代表页眉*/
    font-family:helvetica, arial, sans-serif;
    display:block;
    overflow:hidden;
    width:500px;
    margin:30px;
    border-radius:6px;
    background-color:#ddd;
}
form.stylin_form_search1 { /*包含 label 和 input 的容器*/
    float:right;
    width:200px;
    margin:5px; padding:5px;
}
form.stylin_form_search1 input {
    float:right;
```

```
    width:70px;
    padding:2px 0 3px 5px;
    outline:none; /*去掉默认的突显轮廓线*/
    font-size:.8em; border-color:#eee #ccc #ccc #eee;
    /*针对其他浏览器的厂商前缀*/
    border-radius:10px; -webkit-transition:2s width;
}
form.stylin_form_search1 input:focus {width:200px;}
form.stylin_form_search1 label {display:none;} /*标注是必要的，但不用显示出来*/
```



图 6-19　点击这个小巧的搜索框，它会自动加宽，占位符文本也会被输入的关键字取代

这里的 form 元素是“有宽度的”，而且是向右浮动的。表单内部的 input 也是向右浮动，如图 6-19 上面所示。虽然 label 元素没有显示出来，但它是必须要加的。文本框中的文本由 placeholder 属性生成，只要用户一开始输入，这些占位符文本就会自动隐藏，如图 6-19 下面所示。

运用 CSS3 过渡

在前面的 CSS 中，input 规则将该字段宽度设定为 70 像素，input:focus 规则将该字段宽度修改成了 200 像素。这意味着，在用户单击搜索框让它获得键盘焦点之后，字段宽度会改变。不过，由于现在有了 transition:2s width;声明，字段不会突然扩展到新宽度，而是会用两秒钟时间平滑地伸展到 200 像素。必须注意，CSS3 的过渡声明要放在设定初始状态的规则中。而且，transition 属性需要使用带厂商前缀的形式——这里只示范了带有 WebKit（Safari/Chrome）前缀的属性。下一章，我们还会在一个完整的页面中用到这个搜索表单。另外，下载的本书源代码中还包含它的几个不同版本。要了解有关使用 CSS 过渡的更多信息，请参考附注“CSS3 过渡”。

CSS3 过渡

CSS3 过渡可以让 CSS 属性产生动画效果。平常被某些事件触发时变化很突然的样式，比如鼠标悬停时改变链接颜色，使用过渡后会在指定的时间段内逐渐变化。第一条 CSS 规则设定属性的初始状态和过渡参数。第二条 CSS 规则设定事件发生时属性的目标状态。

在下面这个例子中，用户单击表单输入字段后，输入框的边框颜色会从黑色变化为绿色，过渡周期为两秒钟。

```
input {border-color:black; transition:border-color 2s;}

input:focus {border-color:green;}
```

请注意，使用 transition 属性时要针对所有浏览器添加厂商前缀。

通常，过渡动画是由用户鼠标悬停时的:hover 伪类规则和表单元素获得焦点时的:focus 伪类规则触发的。除此之外，还可以在一个带类名选择符的规则中设定新状态，然后通过 JavaScript（或 jQuery、MooTools 等 JavaScript 库）为元素添加这个类名来触发过渡，添加类名的时机可以是鼠标点击或其他事件发生时。

有五个过渡属性：

- transition-property，过渡的 CSS 属性名，比如 color、width；
- transition-duration，过渡的持续时间，以秒或毫秒设定，比如 2s、500ms；
- transition-timing-function，过渡的调速函数，决定动画效果是否平滑，是先慢后快还是先快后慢，比如 ease-in、ease-out、ease-in-out 或 linear（默认值）；
- transition-delay，过渡开始前的延迟时间，以秒或毫秒设定，比如 1s、200ms；
- transition，过渡的简写属性，例如 transition:color 2s ease-in 1ms;。

很多（并非全部）CSS 属性都可以通过 transition 属性来实现动画效果。至于哪些属性可以实现动画效果，可以参考链接 http://www.w3.org/TR/css3-transitions/#animatable-properties，这个页面是 W3C 对 CSS3 Transitions Module 的官方陈述。另外，下面这篇介绍 CSS3 过渡的文章也非常值得一看：http://www.css3.info/preview/css3-transitions。

6.3　弹出层

弹出层（也叫提示条）指的是在鼠标悬停于某个元素之上时显示的一个界面组件。在页面空间有限的情况下，弹出层是为用户提供更多信息的一种有效手段。而且，

在用户把鼠标移动到关注的元素之上时，显示一个弹出层也是件自然而然的事儿。乍一听，弹出层不是个太复杂的东西。但是，一会儿你就知道我为什么把它放在本章最后才讲了。围绕创建弹出层，我会跟大家讲两个 CSS 中非常强大，但又没多少人真正理解的特性：z-index 属性和动态生成 HTML 元素。我会在接下来的练习中用到三张图片，每张图片都配一个图题。以下就是练习的 HTML 标记，其中使用了 HTML5 新增的 figure 和 figcaption 元素。

```
<figure>
    <img src="images/pink_heels.jpg" alt="pink heels" />
    <figcaption>
        <h3>Pink Platforms</h3>
        <a href="#">More info</a>
    </figcaption>
</figure>
<figure class="click_me">
    <img src="images/leopard_heels.jpg" alt="leopard heels" />
    <figcaption>
        <h3>Leopard Platforms</h3>
        <a href="#">More info</a>
    </figcaption>
</figure>
<figure class="click_me">
    <img src="images/red_heels.jpg" alt="red heels" />
    <figcaption>
        <h3>Red Platforms</h3>
        <a href="#">More info</a>
    </figcaption>
</figure>
```

大家注意，如果要使用 figcaption，必须确保它在 figure 中是唯一的。而且，它要么是第一个子元素，要么是最后一个子元素。首先，我们设定 figure 元素的样式，让它在视觉上成为一个包围图片的盒子。

```
figure {
    width:144px; /*图片盒子宽度*/
    height:153px; /*图片盒子高度*/
    margin:20px 20px; /*图片盒子间距*/
    border:1px solid #666; /*图片边框*/
    position:relative; /*为弹出层提供定位上下文*/
    float:left; /*让图片并排显示*/
}
img {display:block;} /*去掉图片下方的基线空白*/
```

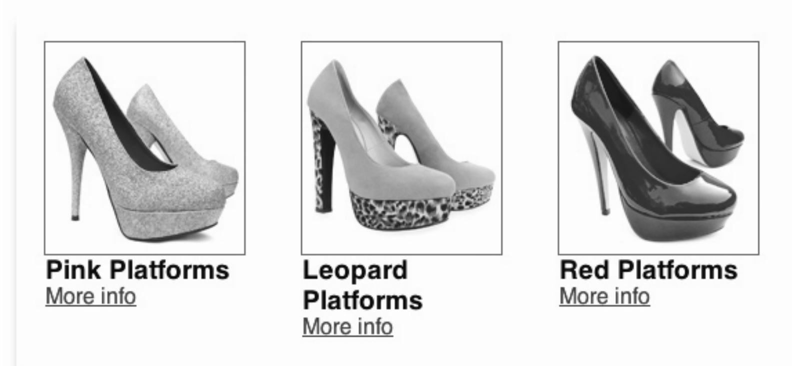


图 6-20　为图片添加了边框，并且让它们并排显示

如图 6-20 所示，figure 元素的边框之内恰好能容纳图片，而且浮动也让它们并排显示在一行。figcaption 元素目前就显示在它默认的位置，但下一步它就会变成我们的弹出层。为什么把图片设定为 display:block 呢？因为图片默认是行内元素，行内元素的定位原则是与文本基线对齐，而不是与它们容器的底边对齐。在把它们放在块级元素内部时（像这里一样），把图片转换为块级元素可以解决这个问题。另外，这里还把 figure 元素设定为相对定位，这样随后就可以把它作为定位上下文来定位 figcaption 了，下面就是相应的样式。

```
figcaption {
    display:none; /*隐藏弹出层*/
    position:absolute; /*相对于容器（图片）定位*/
    left:74%; top:14px; /*把弹出层放到图片右侧*/
    width:130px; /*弹出层宽度*/
    padding:10px; /*弹出层内边距*/
    background:#f2eaea;
    border:3px solid red;
    border-radius:6px;
}
figure:hover figcaption {display:block;} /*鼠标悬停在图片上时显示弹出层*/
figcaption h3 { /*弹出层的内容*/
    font-size:14px;
    color:#666;
    margin-bottom:6px;
}
figcaption a { /*弹出层的内容*/
    display:block;
```

```
    text-decoration:none;
    font-size:12px;
    color:#000;
}
```

这里我们又用到了实现下拉菜单时的技术，即先隐藏弹出层，然后在鼠标悬停时再显示它。为了把弹出层定位在图片盒子的右侧，我们把它的 left 属性设定为 74%。或许有人认为在这里应该使用 right 属性，但那样就是设定图片盒子右边与弹出层右边的相对位置了。

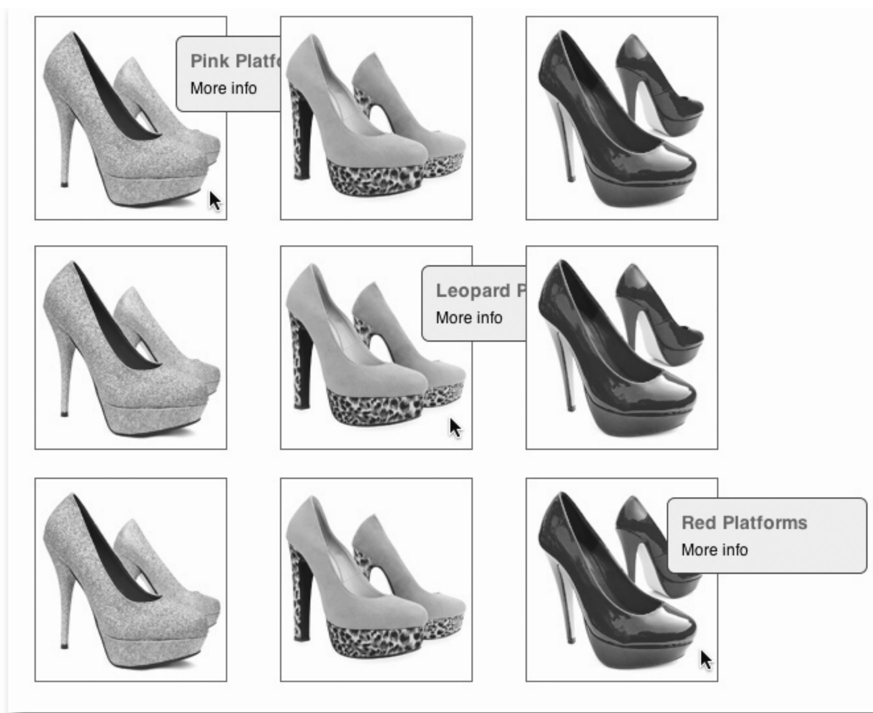


图 6-21 鼠标移动图片上时，会在弹出层中显示图题。可是，前两个截图的弹出层被右边的图片给挡住了

6.3.1 堆叠上下文和z-index

从图 6-21 中的前两个截图来看，前两张图片的弹出层被右边的图片给挡住了一部分。这是由于 figure 元素的堆叠次序导致的。在一个包含多个同辈元素的容器内，就像这里 body 元素的三个 figure 子元素一样，这些同辈元素都会构造一个堆叠上下文。

在这个上下文中，它们的子元素会上下堆叠起来。假如通过定位让前两个 figure 元素重叠，那么你会发现第一个 figure 元素，以及它的堆叠上下文中的所有子元素，都将位于第二个 figure 元素的后面。第一个插图的弹出层属于第一个 figure 元素的堆叠上下文，因此它默认会在第二个 figure 元素的堆叠上下文中所有元素的后面。

CSS 中有一个 z-index 属性，用于控制元素的在堆叠上下文中的次序。换句话说，通过它可以改变元素堆叠时的默认次序。z-index 值较大的元素，在堆叠层次中位于 z-index 值较小的元素上方。z-index 属性的值可以是 0 到任意大的数值；负值也可以，但在某些浏览器中并不可靠。默认情况下，所有堆叠元素的 z-index 的值为 auto，相当于 0。

不过，z-index 只对那些 position 值为 static 之外的元素有效。换句话说，涉及的两个元素必须是 absolute、relative 或 fixed 定位才行。在我们的例子中，弹出层已经是绝对定位了，其位置相对于相对定位的 figure 元素。所以，只要给弹出层应用大于 0 的 z-index 值，就可以让它们在堆叠中处于最高层。这里，我们把 z-index 设定为 0。

```
figure:hover figcaption {display:block; z-index:2;} /*把弹出层放到最前面*/
```



图 6-22　在给弹出层设定了较大的 z-index 属性后，它们就能显示在所有图片之前了

这个例子告诉我们，应该只给悬停状态设定 z-index 属性，而不要试图通过设定个别元素的 z-index 来确保元素不会重叠。只要坚持“悬停时设定 z-index”，你就能放心地摆放图片，而且知道在触发弹出层时，它一定会显示在页面的最前面，如图 6-22 所示。

6.3.2 用CSS创造三角形

现在，我在考虑怎么让弹出层和图片的关系更加明确。嗯，可以给弹出层左边添加一个三角箭头，让它指向图片。这就涉及利用盒子三角形对接的边框来创建三角形了。下面我用一个 div 来演示这个技术。

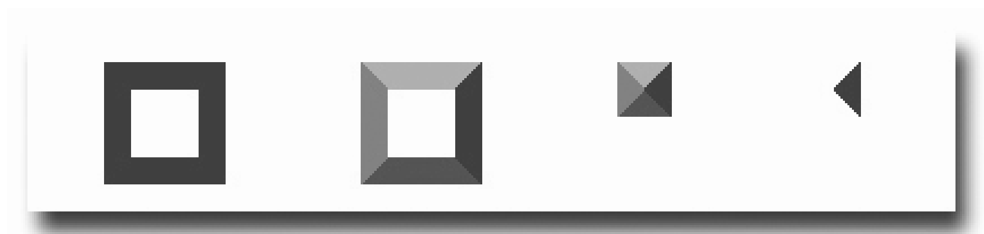


图 6-23 使用宽边框创建三角形

以下是图 6-23 所示效果的代码。

```
div {
    border:12px solid;
    border-color:transparent red transparent transparent;
    height:0px;
    width:0px;
}
```

如图 6-23 所示，通过加宽盒子的边框，将盒子的宽和高都设定为 0，同时将其他三个边框设定为 transparent，就可以用 CSS 造出一个三角形。现在，我把这个技术与::before 伪元素结合起来。大家知道，::before 和::after 这两个伪元素是用于添加文本或图标等少量内容的。不过，完全可以为它们生成的内容设定任何样式，就像给标记中其他元素设定样式一样。在这个例子中，就是要把伪元素生成内容的盒子，通过 CSS 制作成一个三角形，并把它放到弹出层的左边。

```
figcaption::after { /*红色三角形的盒子*/
    content:""; /*需要有内容，这里是一个空字符串*/
    position:absolute; /*相对于弹出层定位*/
    border:12px solid;
    border-color:transparent red transparent transparent;
    right:100%; top:17px; /*相对于盒子边框定位三角形*/
    height:0px; width:0px; /*收缩边框创造三角形*/
}
```

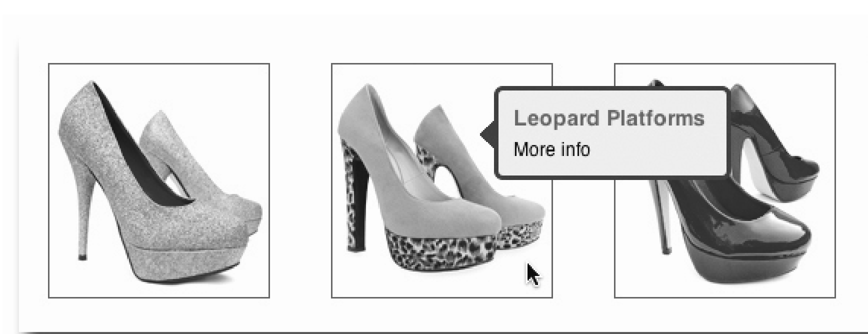


图 6-24　一个小三角形，就从视觉上把弹出层和相关图片联系起来了

好了，代码生成的三角形已经被绝对定位到了弹出层左侧，而弹出层本身又相对于图片定位，如图 6-24 所示。这么 20 来行 CSS，就能显示无数个图片的弹出层！ 谁说编程没有创造性来着？要注意的是，`::before` 伪元素的 `content` 属性中必须有点内容，否则就不能显示生成的元素。因为我们并不真需要什么内容，所以就用一对双引号指定了一个空字符串给它。这个练习好玩吧？肯定比表单好玩多了，是不是？

6.4　小结

本章，我教给了大家一些为流行的 Web 界面组件应用样式的技巧，这些组件包括菜单、表单和弹出层。这一章所有例子的源文件都可以下载到，而且它们的代码改一改就可以用在你的项目中。不过，别忘了：没有什么比徒手从零写出这些组件更能锻炼一个人的 CSS 技能。通过实际操练，你才能深刻理解基本概念，真正搞明白显示、定位、背景，以及其他常用 CSS 特性到底是怎么回事。闲话少说，我们还是快马加鞭吧。下一章，我们就综合运用前一章讲过的布局技术和这一章的界面组件，来制作一个完整的网页。

7 CSS3 实战

这一章我们要写一个完整的网页啦！这个网页要用到第 5 章介绍的页面布局技术，还有第 6 章讲解的界面组件设计技术。不仅如此，我们还会展示很多 CSS3 的新属性，涉及圆角、文字阴影、盒阴影、过渡和变换。这些 CSS3 样式，可以为页面的整体设计增添专业气息。

还是说实话吧，本章这个网页其实就是我在写这本书的时候写的一个页面，也就是本书新版网站中的一个页面。跟以前一样，我要用这个例子来展示对大家自己写项目非常有用的新技术，同时也揭示那些你不小心就可能掉进去的陷阱。

我想先给大家讲一讲怎么规划页面的整体框架，之后再带大家一步一步地为页面每个区域设定 CSS 样式。相信在学习完本章后，你肯定能掌握设计编写一个完整的网页所需的技术，并胜任编写自己新网页的工作。

7.1 规划页面结构

就算写个再简单不过的页面，恐怕都得敲出几百行 HTML 和 CSS。所以，事先规划一下页面结构是非常必要的。记住两条：代码结构要符合逻辑，规划组织要考虑层次。只有这样，才能确保 CSS 与 HTML 标记前后照应、协同一致。后面例子中的代码就是遵循上述规则组织的，我想这应该是编写网页的最佳实践。虽然这种做法需要循规蹈矩，但它的回报也是巨大的。这样一来，不仅能方便地找到与任何标记相

关的 CSS，更能避免样式与标记混乱造成的困惑和时间浪费。最重要的，这样做不仅你自己将来容易理解，而且拿给别人去改的时候，别人也一样能很快看懂。

图 7-1 是本章结束时要完成的页面。



图 7–1　本章要完成的页面布局

HTML结构

着手写 HTML 的时候,怎么才能把画好的设计图转换成恰当的 HTML 元素是个挑战。别急，我有一个好办法。一般来说，这时候你应该有一张用 Photoshop 或 Fireworks 做的设计图，至少也是用铅笔在纸上画出的页面框架图。有一张图，才好跟客户沟通，然后再调整，保证设计的风格还有内容的组织能够满足项目需要。设计图确定之后，就要考虑编写页面代码了。此时，第一步应该先在设计图上画一些半透明的矩形，分别代表主要的 HTML 结构。

HTML 不是由矩形的元素构成的嘛，所以一上来就应该想，怎么把页面布局分成尽可能大的矩形区块。据此，来确定页面标记结构中的顶级元素。然后，再把每个大区块进一步划分成更小的区块，分别对应为结构中的子元素。参见图 7-2，这是我在本章页面设计图上划分区块的结果。

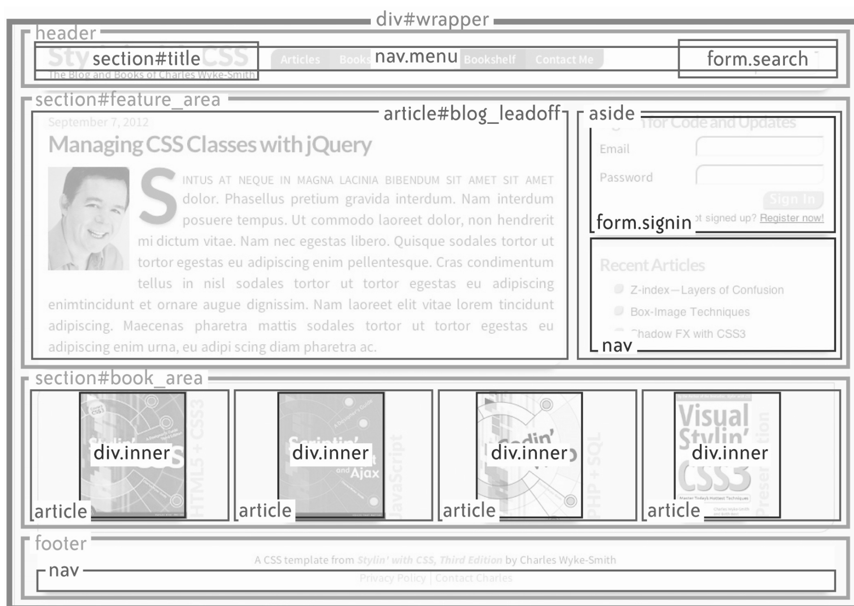


图 7-2　通过观察页面中的矩形区域，可以把页面标记分成三级

从图 7-2 可以看出，页面最外层的粗框是一个外包装（div#wrapper），有了它就可以轻易地设定布局宽度，并且让布局在浏览器窗口内居中。外包装内有 4 个与布局同宽的矩形，它们的框稍细一些，算是布局中的顶级 HTML 元素。这时候，最好想一想给这些元素添加什么类和 ID。不过，header 和 footer 元素是不需要类和 ID 的，因为它们在整个页面中都仅出现了一次。主要是中间两个 section 元素的矩形区块，需要用不同的 ID 来区分，我们就用 feature_area 和 book_area。



有读者可能奇怪，为什么页眉中的导航元素会盖在页面标题和搜索表单上头？因为我已经决定让页面标题和搜索区域采用绝对定位了。这样，导航元素就会忽略它们，从而填满整个页眉。本章后面你就会看到，这样也方便我把它包含的菜单设定为在页面上居中。

然后第二级，也就是图 7-2 中再细一点的那些矩形框。在每个顶级元素中，需要用一个最大的子元素来组织内容。

对 header 而言，其中内容可以分三部分：页面标题在左侧，导航菜单在中间，搜索表单在右侧。在 feature_area 中，左边是博客正文的前几段，右边是一个 aside 元素，包含登录表单和博文链接。这个 aside 区域在下一步还要细分。

再往下的 book_area 需要 4 个容器，因为要放 4 本书。最底下的 footer 包含一点文本和一个结构化的 nav 元素。

接下来是第三级。先说 feature_area aside 吧，要把它分成两个独立的区块，一个表单和一组链接。而在 book_area article 中，每张图片也分别用一个元素包装起来，好为每本书的弹出层提供定位上下文。刚才说的这几个元素构成了结构化标记的第三级。它们的方框在图 7-2 中比第二级的方框要深一点。

虽然实际写起网页来，可能还要额外加一些元素，但页面结构规划到这种程度已经差不多了。下面就是根据以上分析，初步写出的一个标记结构。

```
<div id="wrapper">
    <header><!-- 一级 -->
        <section id="title"><!-- 二级 -->
            <!-- h1 和 h2 -->
        </section>
        <nav><!-- 二级 -->
             <!-- 菜单 -->
        </nav>
        <form> <!-- 二级 -->
            <!-- 搜索框 -->
        </form>
    </header>

    <section id="feature_area"><!-- 一级 -->
        <article id="blog_leadoff"><!-- 二级 -->
            <!-- 博客内容 -->
        </article>
        <aside><!-- 二级 -->
            <form><!-- 三级 -->
                <!-- 登录表单 -->
            </form>
            <nav><!-- 三级 -->
                <!-- 博文链接 -->
            </nav>
        </aside>
    </section>

    <section id="promo_area"><!-- 一级 -->
        <article><!-- 二级 -->
```

```
            <div class="inner"><!-- 三级 -->
                <!-- 图书封面及旋转的文字-->
            </div>
        </article>
        <!-- article 重复 4 次-->
    </section>

    <footer><!-- 一级 -->
        <!-- 页脚文本和 nav 元素 -->
    </footer>
</div> <!-- wrapper 结束 -->
```

仔细看一看这些标记就会知道，布局中嵌套的每一级区块，都对应着标记中嵌套的元素。选择 HTML 元素时，一定要考虑它的语义。比如，菜单列表显然应该放在一个 nav 元素中。

写完结构化标记，可以先通过 body 元素设定页面的字体和颜色，再利用 wrapper 设定布局宽度，并实现布局在页面上居中。

```
body {
    font-family:helvetica, arial, sans-serif;
    background:#efefef;
    margin:0;
}
wrapper {width:980px; margin:0 auto 20px;}
```

好啦，下面就要一步一步地为页面的结构化元素添加内容，同时设定 CSS 样式了。

7.2　页眉

先从页眉（header）开始吧。

```
<header>
    <section id="title">
        <h1>Stylin&#8217; with CSS</h1>
        <h2>The Blog and Books of Charles Wyke-Smith</h2>
    </section>
    <nav class="menu">
         <ul>
```

```
            <li class="choice1"><a href="#">Articles</a></li>
            <li class="choice2"><a href="#">Books</a></li>
            <li class="choice3" ><a href="#">Resources</a></li>
            <li class="choice4"><a href="#">Bookshelf</a></li>
            <li class="choice5"><a href="#">Contact Me</a></li>
         </ul>
    </nav>
    <form class="search" action="#" method="post">
        <label for="user_name">search</label> <!-- 标注的for属性与文本框ID相同 -->
        <input type="text" id="user_name" name="user_name" placeholder=
            "search" />
    </form>
</header>
```

页眉的标记分三部分：页面标题（section#title）、搜索表单（form.search）和导航菜单（nav.menu）。

7.2.1　页面标题

我们把 h1 和 h2 元素定位在了 header 元素的左上角。以下是 CSS。

```
header {
    position:relative; /*为页面标题和搜索表单提供定位上下文*/
    height:70px; /*固定高度，包围绝对定位元素*/
    margin:10px 0;
    background:#fff;
    border-radius:20px 0px 20px 0px; /*顺序：左上、右上、右下、左下*/
    box-shadow:0 12px 8px -9px #555; /*负扩展值把阴影定位到盒子内部*/
    padding:1px; /*防止子元素外边距叠加*/
    }
header section#title {
    position:absolute;
    width:300px; /*宽到足以不让文本折行*/
    height:65px; /*高到足以容纳两行文本*/
    left:0px; /*左上角定位*/
    top:0;
    }
header h1 {
    padding:9px 12px 0;
    font-family:'Lato', helvetica, sans-serif;
```

```
    font-weight:900;
    font-size:2.2em;
    line-height:1;
    letter-spacing:-.025em;
    color:#4eb8ea;
    }
header h2 {
    padding:0px 12px;
    font-family:"Source Sans Pro", helvetica, sans-serif;
    font-weight:400; /*设定下载字体的粗细*/
    font-size:.9em; line-height:1;
    letter-spacing:-.025em;
    color:#333;
    }
```

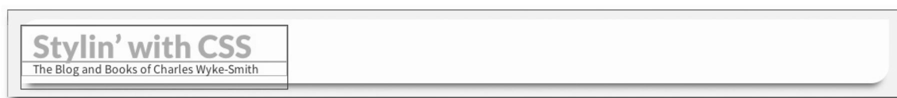


图 7-3 为 h1 和 h2 设定了样式，临时添加的边框显示了它们的位置

这里要注意的第一件事，就是明确地为 header 设定了固定高度。在前面的例子中，我们一直提倡让内容本身决定结构化元素的高度，让高度随内容增多而扩展。可是在这里，由于 header 包含的是绝对定位的元素，该元素不会影响父元素高度，所以必须明确设定这个高度。header 的内容是极少会改动的，因此未来也不大可能出现内容溢出 header 的情况。

其次，就是我把“两个圆角，两个方角”的样式，应用给了 header 及页面中的很多其他元素。这种不张扬但又与众不同的效果，为页面赋予了独特的外观。要了解如何为 HTML 元素盒子应用圆角，请参考附注栏“弧形角”。

接着，给 section#title 设定了绝对定位，在确定其文本内容的大小和内边距之后，又给它设定了足以包含这些文本的固定宽度和高度。

对于这里标题的文本，我们使用了 Google Web 字体 Lato。而在设定 Google Web 字体粗细的时候，font-weight 属性的值使用是数字值。这跟使用系统安装的字体不一样，对于系统安装的字体，浏览器通常只能显示 font-weight 的 normal 和 bold 两个值。要了解有关 Google Web 字体的更多信息，请回顾第 4 章。

另外一处别出心裁的设计，就是盒子的阴影。我们的 CSS 会控制这些阴影只显示在盒子底部，而且比盒子自身窄一些。这样，盒子就好像在页面上悬空一样。前面代码中加粗了相应声明，有关盒阴影的更多信息请参考附注栏“盒阴影”。

页面标题就位之后，接下来我们再采取类似的方法把搜索框定位到页眉区域右侧。

弧形角

圆角效果可是几年前 Web 2.0 网页的招牌式设计。当时，实现简单的圆角要用复杂的 JavaScript，或者得用嵌套的 DIV 和丝毫不差的图片定位。而现在，则只要一行 CSS 就能搞定了。

最简单语法形式如下。

```
border-radius:10px;
```

这样，盒子的四角都会变成半径为 10 像素的圆角。

如果要单独设定每个角的半径，也可以在上面的简写属性中按顺序依次指定。只不过，现在指定的是角而不是边，所以“上、右、下、左”的顺序就不适用了，而是要改用“左上、右上、右下、左下”。

另外，也可以像下面这样分别设定水平和垂直半径：

```
border-radius:10px / 20px;
```

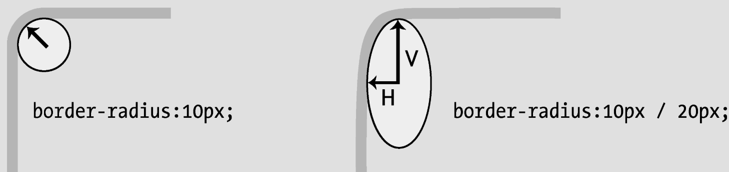


图 7-4　前面两个 CSS 声明的效果示例

图 7-4 也说明了为什么管这种效果叫弧形角了，即我们设定的值表示的是位于角内部的圆形或椭圆形的半径。

如果你想给每个角都设定不同的水平和垂直半径，写法如下：

```
border-radius:10px 6px 4px 12px / 20px 12px 8px 24px; /* 4 个水平值，4 个垂直值 */
```

最后提醒大家，弧形角不一定要通过边框才能显示出来。大家看到图 7-1 中的菜单了吗，菜单的圆角就是通过元素的背景色而非边框显现出来的。

盒阴影

HTML 元素的阴影，同样是 CSS3 被广泛实现之前很难做出来的一个效果。当时，要给元素添加阴影效果，必须用图片和 DIV 配合，还要耐心地调整，而现在则只要一行 CSS 声明即可。

最简单语法形式如下。

```
box-shadow:4px 4px 5px 8px #aaa inset;
```

box-shadow 属性的这几值分别代表：水平偏移量、垂直偏移量、模糊量、扩展量、颜色、阴影位于边框内部（默认位于边框外部，即 outset）。

最低限度，必须设定水平偏移量、垂直偏移量和颜色，这样会得到一个与元素宽度和高度大小一致且为指定颜色的硬边阴影。如果水平和垂直偏移量是负值，阴影就会出现在元素左上方。inset 关键字会把阴影放到盒子内部。另外，box-shadow 还支持多个阴影声明。图 7-5 展示了一些阴影示例。

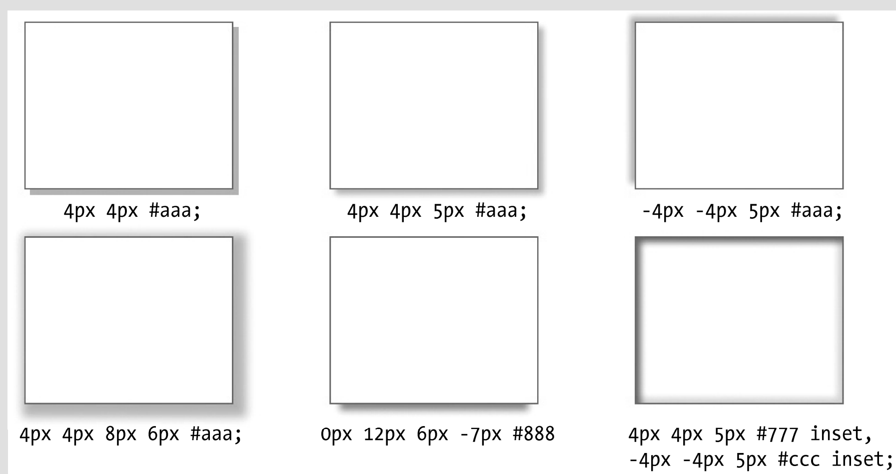


图 7-5　使用正、负偏移量，实现了各种不同的阴影效果

7.2.2　搜索表单

还是先温习一下标记吧。

```
<form class="search" action="#" method="post">
    <label for="user_name">search</label>
    <input type="text" id="user_name" name="user_name" placeholder="search" />
</form>
```

以下其实就是上一章搜索表单示例的 CSS。唯一明显不同的，就是表单在页眉中定位的方式。

```
form.search {
    position:absolute; width:150px; /*宽到足以容纳扩展后的搜索框*/
    top:23px; right:20px; /*相对于页眉右上角定位*/
    }
.search input {
    float:right; width:70px;
    padding:2px 0 3px 5px;
    border-radius:10px 0px 10px 0px;
    font-family:"Source Sans Pro", helvetica, sans-serif;
    font-weight: 400;
    font-size:1em;
    color:#888;
    outline:none; /*去掉默认的轮廓线*/
    -webkit-transition:2s width; /*搜索框过渡动画，别忘了带其他厂商前缀的属性*/
    }
.search input:focus { width:140px;}/*在获得焦点时扩展到这么宽*/
.search label{display:none;}
form.search input {background-color:#fff;}
form.search input::-webkit-input-placeholder {color:#ccc;}
```

上面加粗的代码表明，我们为表单元素设定了宽度，并将它的右边相对于页眉的右边定位。如图 7-6 中临时添加的边框所示，我们为表单设定了足够的宽度，让它能够容纳扩展后的输入框。关于搜索表单我不想在这儿说太多，因为上一章已经详细地讲过了。但有一点我想强调，就是——至少在 WebKit 浏览器中——可以给占位符文本设定不同样式，让它有别于用户输入的文本。如何设定请参见前面代码最后一行。

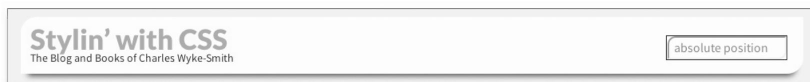


图 7-6　搜索框已经扩展到最大宽度，其边框是临时添加的

7.2.3　菜单

现在，页眉两端有了两个绝对定位的元素，这两个元素已经离开了常规文档流。接下来，我必须让菜单在它们之间居中。这里的菜单使用的也是第 6 章的菜单，只是少量修改了几个地方而已。



为节省版面，这里没有列出生成下拉菜单的所有嵌套的标记。如果你想回顾如何用 CSS 制作下拉菜单，请参考第 6 章。

```
<nav class="menu">
     <ul>
        <li class="choice1"><a href="#">Articles</a></li>
        <li class="choice2"><a href="#">Books</a>
        <!-- 更多菜单项 -->
</nav>
```

包含在 nav 元素中的仍然是普通的链接列表，但现在每个列表项都有了不同的类，方便我们设定颜色。

以下是 CSS。

```
nav.menu  {
    margin:19px auto;
    padding:0;
    text-align:center;  /*在容器内居中菜单*/
    font-size:.8em;
    }
nav.menu > ul { display:inline-block;} /*收缩包紧列表项*/
nav.menu li {
    float:left; /* 让菜单项水平排列*/
    list-style-type:none; /*去掉默认的项目符号*/
    position:relative; /*为子列表提供定位上下文*/
    }
nav.menu li a {
    display:block; /*让链接填满列表项*/
    padding:.25em .8em;
    font-family:"Source Sans Pro", helvetica, sans-serif;
    font-style: normal;
    font-weight:600;
    font-size:1.2em;
    text-align:left;
    color:#fff;
    text-decoration:none; /*去掉链接的下划线*/
    -webkit-font-smoothing: antialiased; /*在 WebKit 浏览器中平滑字体*/
    }
nav.menu li.choice1 a {background:#f58c21;}
nav.menu li.choice2 a {background:#4eb8ea;}
nav.menu li.choice3 a {background:#d6e636;}
```

```
nav.menu li.choice4 a {background:#ee4c98;}
nav.menu li.choice5 a {background:#f58c21;}
nav.menu li:hover > a {
    color:#555;
    border-color:#fff;
    border:0;
}
nav.menu li:last-child a  {border-bottom-right-radius:10px;}
nav.menu li:first-child a  {border-top-left-radius:10px;}
```

页面标题和搜索表单是绝对定位，因而离开了常规文档流。换句话说，nav 元素就会当它们不存在，而在水平方向上扩展填充父元素 header（图 7-7）。这样我就可以让菜单在页面上居中。下面就以此为例，再讲一些使用 CSS 居中元素的技巧。

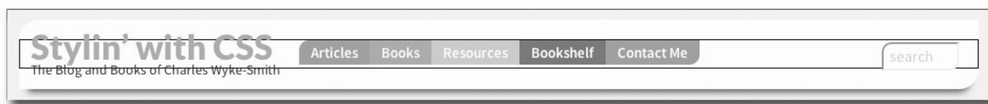


图 7-7　菜单在页眉内居中了。由于左右两端的页面标题和搜索表单都因绝对定位而脱离了文档流，所以 nav 元素是与布局同宽的（如临时添加的边框所示）

居中没有宽度的元素

在一个元素内居中另一个元素有时候会很困难。对于常规、静态定位的元素，可以让它向左或向右浮动，或者使用 text-align 属性让它在父元素内居左、居右或居中。还可以利用自动外边距（margin:0 auto）来居中元素。这些方法的问题在于，要居中的元素必须是有宽度的。像这里用于构成菜单的 HTML 列表，它可能是根据数据库信息动态生成的，或者说将来有可能手工编辑，总之你不可能提前设定它的宽度。我收到很多邮件，都是询问怎么在容器内部居中菜单的。下面也算是统一回答吧，我们来看看怎么居中一个没有设定宽度的元素。

在 display 属性的值中，inline-block 具有一些特殊的混合行为。正如它的名字所暗示的，它既有块级元素的特点，也有行内元素的行为。从块级元素角度说，可以为它设定外边距和内边距，也可以通过它简便而有效地包围其他块级元素。从行内元素角度看，它会收缩包裹自己的内容，而不是扩展填充父元素。换句话说，inline-block 元素的宽度始终等于其内容宽度。这种元素还有一个特点，就是可以包围浮动元素。不过，这种元素也有一个问题，即不能给它的外边距设定 auto 值——而这恰恰又是在更大的容器内居中元素的最简单方法。

解决方案就是为要居中元素的父元素（这里的 nav）应用 text-align:center，为要居中的元素（这里的 ul）设定 display:inline-block，让它包含列表项。这样设定就可以得到我们想要的结果：没有固定宽度的元素也能在其父元素内居中。如前面代码开头加粗的 CSS 声明所示，我们就是这么做的。现在菜单完美居中了，因为其父元素 nav 忽略了两端绝对定位的元素，扩展到了与 header 同宽。

为了演示这个技巧有多好，我们来去掉一个菜单项。

如图 7-8 所示，去掉最后一个菜单项之后，整个菜单照样居中。对于不同用户会看到不同菜单（比如，区别是否为注册会员）的动态站点来说，这个技巧的确太有用了。还有一个地方要特别留意，就是我没有直接为第一和第五个菜单项添加圆角效果，否则将来如果增减菜单项就会出问题。如前面代码末尾加粗的声明所示，为了让 CSS 适应菜单项的动态变化，我们把圆角效果应用给了:first-child 和 last-child 元素。这样一来，第五个菜单项被删掉后，第四个菜单项就会成为最后一个子元素，从而取得应用给:last-child 的圆角效果。

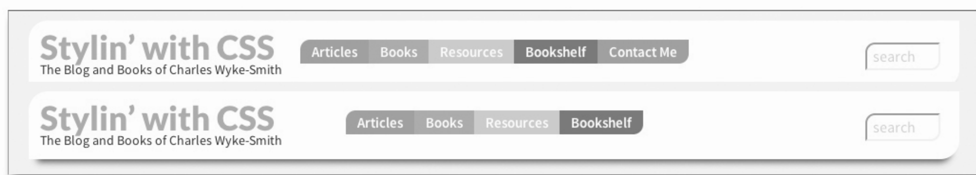


图 7–8　即使去掉菜单项，菜单整体上仍然居中

添加下拉菜单

下面，利用添加下拉菜单的机会，我来给大家介绍另一种 CSS 过渡效果。

```
nav.menu li ul {
    opacity:0; visibility:hidden; /* 隐藏下拉菜单*/
    position:absolute; /*相对于父菜单定位*/
    width:12em; /*下拉菜单宽度*/
    left:0;   /*左边与父菜单项左边对齐*/
    top:100%; /*顶边与父菜单项底边对齐*/
    -webkit-transition: 1s all; /*设定过渡效果*/
    -moz-transition: 1s all;
    transition: 1s all;
    }
nav.menu li:hover > ul {
```

```
    opacity:1; visibility:visible; /*两个属性都会产生过渡动画*/
    }
nav.menu li li {
    float:none; /*去掉继承的浮动，让菜单项上下堆叠*/
    }
nav.menu li li:first-child a {border-radius:0;}
nav.menu li li:last-child a {border-bottom-left-radius:10px;}
.no-csstransitions nav.menu li ul { /*针对不支持过渡的浏览器*/
    visibility:visible; /*覆盖过渡声明*/
    opacity:1;  /*覆盖过渡声明*/
    display:none; /*如果不支持过渡，就直接隐藏菜单*/
    }
.no-csstransitions nav.menu li:hover > ul {
    display:block; /*在父菜单项悬停时显示菜单*/
    }
```

这里的代码也是第 6 章相应例子中的代码，所以如果有什么问题，请大家参考第 6 章以及前面 CSS 中的注释。不过，关于这里的菜单，我还想跟大家解释三点。

首先，下拉菜单会继承圆角样式。但对于下拉菜单来说，我们不想像顶级菜单一样把该样式应用到对角上，而只想应用给最后一个菜单项下面的两个角。所以，我们覆盖了继承的左上角的圆角边框，又给左下角添加了圆角边框样式。当然，右下角的圆角边框仍然是继承来的（图 7-9）。

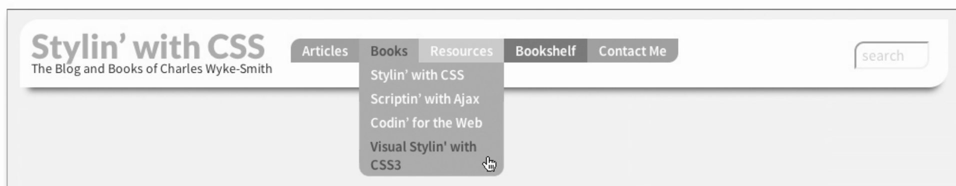


图 7–9　添加了下拉菜单

其次，代码注释里也提到了，我们为下拉菜单应用了不透明度过渡，让它产生淡入效果。我一开始只使用了 `opacity` 属性，把它的起始值设定为 0（透明，不可见），终止值设定为 1（不透明，完全可见）。这样确实可以在鼠标悬停时让下拉菜单淡入淡出，但即使不可见的时候，它也会在那里。换句话说，鼠标移动到顶级菜单项下方，甚至还没有到那一项时，其下拉菜单就会淡入。后来，我又尝试添加 `display:none` 和 `display:block`，即在非悬停状态完全去掉下拉菜单。这倒是解决了下拉菜单“占位”的问题，但过渡效果就没有了，下拉菜单会突然出现、消失。然后，我又放弃

display，改为在悬停状态关掉 visibility 属性，同时过渡 opacity。结果是下拉菜单淡入，但会突然消失。最后，还是同时过渡 opacity 和 visibility 属性解决了问题——菜单在透明时消失无形，淡入和淡出都正常了。说了这么多，主要是想告诉大家，有时候要得到某个结果，总免不了要多试验几次。不过我还是很欣慰的，因为我的试验能让大家节省好几个小时。大家就不用跟我客气啦！

第三，我还利用这个机会向大家展示了 Modernizr 在实际中的应用，通过它为那些不支持 CSS3 过渡的浏览器提供了后备 CSS 声明。具体来说，在不支持过渡的浏览器中，Modernizr 会在页面加载后为顶级 HTML 标签添加 no-csstransitions 类。我们可以把这个类名放到选择符里，写几条只有不理解 CSS 过渡的浏览器才会用的规则。在这些规则里，去掉在支持过渡的浏览器中控制菜单的 visibility 和 opacity 声明，为能力欠缺的浏览器提供基本的 display:none 和 display:block 声明，实现第 6 章那种简单的显示和隐藏菜单的效果。

到现在为止，header 部分已经完事了。下一节，我们学习怎么为功能区设定样式，这个区域包括博客文章、登录表单和博文链接。

垂直居中

用 CSS 实现垂直居中也不简单。如果你想在一个固定高度的元素内垂直居中一行文本，可以把这一行文本的 line-height 设定为该元素的高度。假设元素高度为 300 像素，可以这样写：

```
text-align:center; /*水平居中*/
line-height:300px; /*垂直居中：行高=容器高度*/
```

如果想垂直居中其他元素，比如图片，可以将容器的 display 属性设定为 table-row，再设定（只对单元格有效的）vertical-align 属性为 middle，比如：

```
display:table-cell; /*借用表格的行为*/
vertical-align:middle; /*垂直居中*/
text-align:center; /*水平居中*/
```

尽管这些方案都不够自然，但 CSS 没提供什么垂直定位元素的属性，也就只能这么将就了。

7.3　专题区

页面的专题区主要用于展示最新博客文章的前几段文字，同时右侧还有一小块区域，显示登录表单和最近博文的一组链接。以下就是 section#feature_area 元素的 HTML 标记。

```
<section id="feature_area">
    <article id="blog_leadoff">
        <div class="inner">
            <h4>September 7, 2012</h4>
            <a href="#"><h3>Managing CSS Classes with jQuery</h3></a>
            <img src="images/charles_wyke-smith.jpg" alt="Charles Wyke-Smith
                photo" />
            <p>Sintus at neque in magna...</p>
        </div>
    </article>
    <aside>
        <form  autocomplete="off" class="signin"
               action="process_form.php" method="post"> <!-- 必要的<form>标签 -->
        <fieldset> <!-- 作为表单控件的容器 -->
            <!-- 控件组的标题 -->
            <legend><span>Sign In for Code and Updates</span></legend>
            <section> <!-- 用于为控件、标注和说明添加样式的外包装 -->
                <!-- 与控件 ID 同名的 for 属性将标注与控件关联起来 -->
                <label for="email">Email</label>
                <!-- type 属性的 text 值表明这是文本框 -->
                <input type="text" id="email" name="email" />
            </section>
            <section>
                <label for="password">Password</label>
                <input type="password" id="password" name="password"
                       maxlength="20" /> <!-- 密码框中的字符显示为掩码 -->
                <p class="direction">Wrong user name or password</p>
            </section>
            <section> <!-- 提交按钮 -->
                <input type="submit" value="Sign In" />
                <p class="signup">Not signed up? <a href="#">Register now!
                    </a></p>
            </section>
        </fieldset>
```

```
        </form>
        <nav>
            <h3>Recent Articles</h3>
            <!-- 博文链接 -->
        </nav>
    </aside>
</section>
```

这个 section 元素与容器同宽，我们要把它包含的 article 和 aside 元素浮动为并列显示。

```
section#feature_area {
    overflow:hidden; /*包围浮动的子元素*/
    margin:16px 0 0; /*在页眉与专题区之间留出间隙*/
    padding:0 0 10px;
}
section#feature_area article {float:left; width:66%;}
section#feature_area aside {float:right; width:34%;}
```

这样就在容器里创建了两栏。注意，两栏的宽度是用百分比值设定的，这是因为这个页面还要考虑适应不同的设备，包括平板电脑和智能手机。具体内容将在下一章讲解。为此，两栏就用容器宽度的百分比来设定了。

此时，应该在 article 元素内部添加一个内部 div（前面 HTML 标记中加粗了），以便围绕内容设定圆角边框。之后，就是 article 区域的样式。

```
section#feature_area article .inner {/*带圆角和阴影的容器*/
    padding:12px;
    background:#fff;
    border-radius:20px 0;
    box-shadow:0 12px 8px -9px #555;
    }
section#feature_area article a {text-decoration:none;}/*博文标题链接*/
section#feature_area article img { /*照片*/
    float:left;
    padding:0 10px 10px 0;
    }
section#feature_area article h4 {  /*日期*/
    font-family:"Source Sans Pro", helvetica, sans-serif;
    font-style:normal;
    font-weight:400;
    font-size:1em;
```

```
    color:#f58c21;
    letter-spacing:-.025em;
    }
section#feature_area article h3 { /*博文标题*/
    font-family:Lato, helvetica, sans-serif;
    font-style:normal;
    font-weight:700;
    font-size:1.75em;
    color:#555;
    margin:0px 0 12px 0px;
    letter-spacing:-.05em;
    }
section#feature_area article#blog_leadoff p { /*博文内容*/
    font-family:"Source Sans Pro", helvetica, sans-serif;
    font-style: normal;
    font-weight:400;
    font-size:1.1em;
    line-height:1.5em;
    color:#616161;
    margin:0 0px;
    text-align:justify;
    }
section#feature_area article#blog_leadoff p::first-letter { /*首字母下沉*/
    font-family:Lato, helvetica, sans-serif;
    font-style: normal;
    font-weight:700;
    font-size:4.5em;
    float:left;
    margin:.05em .05em 0 0;
    line-height:0.6;
    text-shadow:1px 3px 3px #ccc; /*IE10 及以上版本支持文本阴影*/
    }
section#feature_area article#blog_leadoff p::first-line {  /*首行小型大写字母*/
    font-variant:small-caps;
    font-size:1.2em;
    }
section#feature_area aside { /*右栏*/
    width:34%;
    float:right;
    }
```

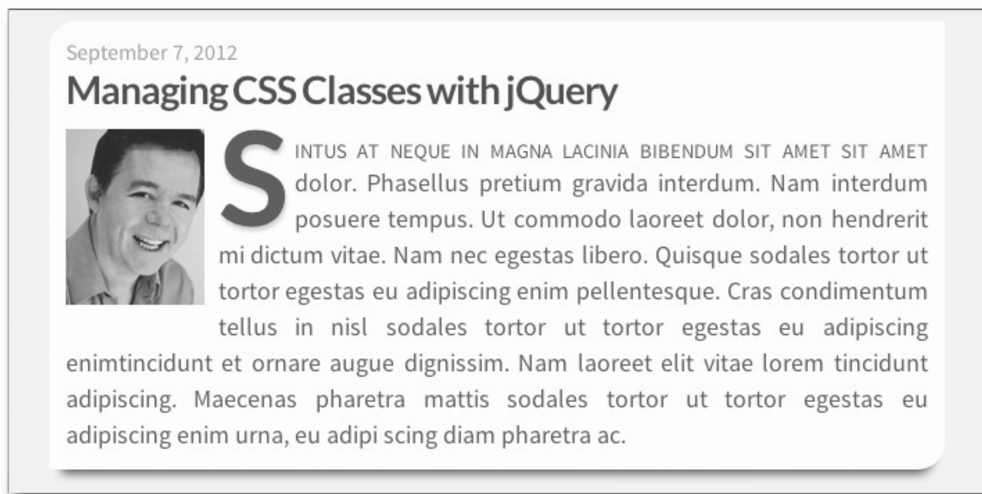


图 7-10 section#feature_area 中的 aticle 元素在设定了样式之后的效果

这些样式涵盖了本书前几章介绍一些元素。值得一提的是包含 h3 的 a 元素。要是在以前，行内元素包含块级元素可是大忌。但从 HTML5 开始，a 元素可以包含任何元素，这当然为把任何元素转换成可以点击的链接提供了方便。

如图 7-10 所示，照片浮动到左侧，所以文本绕排。然后我们采用第 4 章讨论的首字母下沉和首行大型小写样式，为版式添加了一点趣味性，也让字形过渡更加自然。另外，我们还给放大的第一个字母添加了文本阴影，让它看起来像是悬浮在页面上。好了，下面该讲为表单添加样式了。

文本阴影

文本阴影与本章前头讲过的盒阴影很相似，它的语法如下：

```
text-shadow:4px 4px 5px #aaa;
```

text-shadow 这几值的含义按顺序分别是：水平偏移量、垂直偏移量、模糊量和颜色。与盒阴影不同的是，文本阴影没有扩展量。最低限度，你得提供水平偏移量、垂直偏移量和颜色值。如果水平和垂直偏移量是负值，阴影就会出现在文本左上方。另外，text-shadow 还支持多个阴影声明。图 7-11 展示了一些阴影示例。要了解文本阴影更高级的应用技巧，请大家参考我的电子书 *Visual Stylin' with CSS3*。

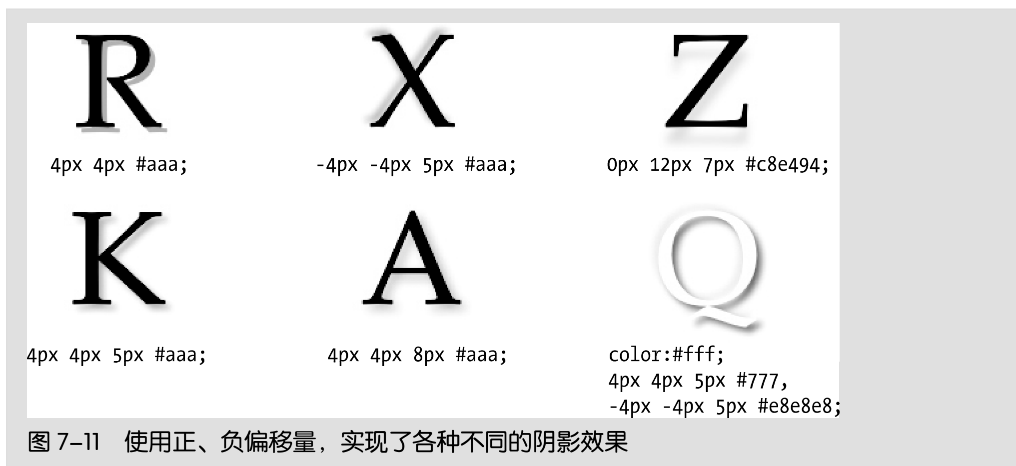


图 7-11　使用正、负偏移量，实现了各种不同的阴影效果

7.3.1　登录表单

我要求读者必须注册才能下载本书示例代码，以便提供更新并与大家保持联系。在这个主页上，提供了一个登录表单和一个指向注册表单的链接，注册表单是为第一次访问网站的读者准备的。登录表单的标记与第 6 章介绍的表单采用相同的结构方式，以下是其 HTML。

```
<form  autocomplete="off" class="signin"
       action="process_form.php" method="post"> <!-- 必要的<form>标签 -->
<fieldset>
    <!-- 控件组的标题 -->
    <legend><span>Sign In for Code and Updates</span></legend>
    <section> <!-- 电子邮件 -->
        <!-- 与控件 ID 同名的 for 属性将标注与控件关联起来 -->
        <label for="email">Email</label>
        <!-- type 属性的 text 值表明这是文本框 -->
        <input type="text" id="email" name="email" />
    </section> <!-- 密码 -->
    <section>
        <label for="password">Password</label>
        <input type="password" id="password" name="password"
               maxlength="20" />
        <!-- 只有添加 error 类才会显示 -->
        <p class="direction">Wrong user name or password</p>
    </section>
```

```
    <section> <!-- 提交按钮 -->
        <input type="submit" value="Sign In" />
        <p class="signup">Not signed up? <a href="#">Register now!</a></p>
    </section>
</fieldset>
</form>
```

下面是我从第 6 章表单的 CSS 中拿来的规则，并针对这个表单修改后的代码。

```
form.signin {
    width:19em; /*表单的整体宽度*/
    float:right;
    background:#fff;
    border-radius:10px 0 10px 0;
    box-shadow: 0 12px 8px -9px #555;
    }
.signin fieldset {  border:0; margin:10px 14px;}/*去掉默认的边框*/
.signin legend span {
    font-family:Lato, helvetica, sans-serif;
    font-weight:700; font-size:1.3em; line-height:1.1em;
    color:#4eb8ea;
    letter-spacing:-.05em;
    }
.signin section {
    overflow:hidden; /*包围控件和标注*/
    padding:.25em 0; /*表单元素的间距*/
}
.signin section label {
    font-family:"Source Sans Pro", helvetica, sans-serif;
    font-weight:400;
    float:left;
    width:5em; /*标注栏的宽度*/
    margin:.5em .3em 0 0; /*外边距保持文本与控件的间距*/
    line-height:1.1;
    color:#555;
    }
.signin section input {
    float:right;
    width:10.5em; /*控件栏的宽度*/
    margin:.2em 0 0 .5em;
    padding:3px 10px 2px; /*输入文本与控件的间距*/
    color:#555;
```

```
    font-size:.8em;
    outline:none; /*去掉默认的轮廓线*/
    border-radius:10px 0 10px 0;
    }
input:-webkit-autofill { color:#fff !important; } /*去掉 WebKit 默认的黄色背景*/
.signin section input[type=submit] {
    float:right; /*将按钮与控件右边对齐*/
    width:auto; /*重设按钮宽度*/
    margin:0 2px 3px 0;
    padding:0px 8px 3px;
    font-size:1em;
    font-weight:800;
    color:#fff;
    border:none;
    background-color:#d6e636;
    box-shadow:1px 1px 2px #888;
    }
.signin section p{ /*内容为"not signed up?"*/
    float:right;
    clear:both;
    margin:.2em 0 0;
    text-align:right;
    font-size:.8em;
    line-height:1;
    color:#555;
    }
.signin section p a { color:#333; }/*到注册表单的链接*/
.signin section p a:hover {
    color:#777;
    text-decoration:none;
}
.signin section p.direction.error { /*错误消息*/
    display:block;
    color:#f00;  /*添加 error 类后，把说明文字变成红色*/
    }
.signin section p.direction { display:none; } /*隐藏错误消息*/
```

无论多简单，表单所需的代码总是很多！好在，这里的代码大部分都很直观，配合注释看基本都能理解。我想提一下错误消息，它一开始是隐藏的，只有必要时才会显示（如图 7-12 所示）。要显示错误消息，只要给（已经有了 direction 类的）p 元素再添加一个 error 类即可。不过，这个类是要通过验证表单的代码来添加的。作

为负责网站 UI 的人，添加这种平时隐藏的 HTML 元素和显示它的 CSS 是你的事儿。至于什么时候添加 `error` 类显示错误消息，那就是开发团队的事儿了。



图 7-12 登录表单及显示错误消息时的效果

7.3.2 博文链接

表单下面是博文链接。与以往一样，我使用了无序列表来组织链接。

```
<nav>
    <h3>Recent Articles</h3>
    <ul>
        <li><a href="#">Z-index&mdash;Layers of Confusion</a></li>
        <li><a href="#">Box-Image Techniques</a></li>
        <li><a href="#">Shadow FX with CSS3</a></li>
    </ul>
</nav>
```

CSS 如下。

```
section#feature_area nav {
    width:19em; /*容器整体宽度*/
    float:right;/*与区域右边对齐*/
    margin:15px 0 0; /*上方间距*/
    padding:.6em 0em .75em; /*链接上下的间距*/
    background:#fff;
    border-radius:10px 0 10px 0;
    box-shadow: 0 12px 8px -9px #555;
    }
#feature_area nav h3 {
    padding:0 14px 0; /*标题左右的空间*/
    font-family:Lato, helvetica, sans-serif;
```

```
    font-weight:700;
    font-size:1.3em;
    text-align:left;
    color:#aaa;
    letter-spacing:-.05em;
    }
#feature_area nav ul { margin:0em 0 0 20px; }
#feature_area nav li {
    padding:.7em 0 0 2em;
    position:relative; /*项目符号的定位上下文*/
    list-style-type:none
    }
#feature_area nav li:before { /*定制项目符号*/
    content:""; /*用空字符串，因为不需要实际内容*/
    position:absolute; /*相对于列表项定位*/
    height:10px; /*项目符号大小*/
    width:10px;
    left:12px; /*定位项目符号*/
    top:12px;
    border-radius:5px 0 5px 0; /*项目符号形状*/
    background-color:#d6e636; /*项目符号颜色*/
    box-shadow:1px 1px 2px #888;
    }
#feature_area nav li a {
    display:block; /*链接与列表项同宽*/
    text-decoration:none; /*去掉默认的下划线*/
    font-size:.9em;
    color:#616161;
    }
#feature_area nav li a:hover { color:#000; }
```

添加了样式之后的链接区就在登录表单正下方。它们都是 aside 元素的子元素，而 aside 元素通过浮动与 article 元素并列在一行。

与浮动 form 元素一样，浮动 nav 元素可以让它直接定位在表单下方。列表的其他部分没有什么特别，但为了体现“两个圆角，两个方角”的设计风格，我们要重新定制项目符号。为此，我们抛弃用图片作为列表项记号的常规做法，使用::before 创建了 10 像素见方的伪元素，并将其两角变成圆角。而一个像素的小阴影则让这些项目符号产生了跳脱页面的假象。



图 7-13　完成博文链接，aside 元素的样式就写好了

7.4　图书区

接下来，有四本图书封面的展示区位于页面底部。为了给这一部分增添趣味性，我们要使用弹出层和旋转文本。先看 HTML 标记。

```
<section id="book_area">
    <article class="left">
        <div class="inner">
            <h3>HTML5 + CSS3</h3><!-- 要旋转的文字 -->
            <img src="images/stylin_cover.png" alt="Stylin' with CSS cover" />
            <aside> <!-- 弹出层 -->
                <ol>
                    <li><a href="#">Download the Code</a></li>
                    <li><a href="#">Table of Contents</a></li>
                    <li><a href="#">Buy this Book</a></li>
                </ol>
            </aside>
        </div>
    </article>
    <!-- 另外三本图书的标记也一样 -->
</section>
```

怎么样，HTML 代码极其简单，同样的标记重复四次，就是四本书。下面来看看相关的 CSS 吧，先看看布局和旋转文本的样式，然后再看弹出层的样式。

```
section#book_area   { /*与布局同宽*/
    clear:both;
    border-radius:20px 0px 20px 0px;
    border:1px solid #f58c21;
    margin:8px 0 16px; /*上下间距*/
    overflow:hidden;
    }
#book_area  article { /*四本书四栏*/
    float:left;
    width:25%;
    padding:10px 0;
    background:none;
    }
#book_area article .inner { /*封面外包装*/
    position:relative; /*为弹出层提供定位上下文*/
    width:140px; /*包装每一本书*/
    margin:0 auto; /*在各自 article 元素内居中每一本书*/
    }
#book_area  .inner h3 { /*旋转文字*/
    position:absolute;
    width:160px;
    left:112%; bottom:5px; /*把文字定位在图书右侧*/
    transform:rotate(-90deg); /*旋转文字需要使用带厂商前缀的属性*/
    transform-origin:left bottom; /*设定旋转中心点，需要带厂商前缀的属性*/
    color:#ccc;
    font-size:1.4em;
    font-family:Lato, helvetica, sans-serif;
    font-style:normal;
    font-weight:900;
    text-align:left;
    }
/*较窄的封面需要不同的偏移量*/
#book_area article.right:last-child h3 { left:85%; }
#book_area article img { box-shadow: 0 12px 8px -9px #555; }/*封面阴影*/
```

在与布局同宽的 section#book_area 元素中，我们浮动了四个 article 元素，每个元素的宽度都设定为 25%。在这四个 article 中，分别使用一个有宽度的内部 div 来包

含图书封面。这样就可以为图书封面之间添加适当的间距，如图 7-14 所示。接下来要做成弹出层的 aside 元素目前还处于隐藏状态。



图 7–14 图书区的每本书封面的右侧都有自己的描述性标题

旋转文字是用 CSS3 的 transform 属性的两个函数实现的。第一个函数是 transform-origin，用于把旋转变换的原点设定为 h3 元素盒子的左下角。这里的原点指的是旋转的中心点，就好像我在那个地方插进去一根大头针一样。然后用 transform 的 rotate 函数把 h3 元素旋转 90 度，最后再将它们向上移动 5 像素。关于 transform 属性的更多介绍，请参考附注栏“CSS3 变换”。如果觉得不解渴，可以购买我的电子书 *Visual Stylin' with CSS* :)

CSS3 变换

如果你用过 Adobe Illustrator 或 Fireworks 等平面图形设计软件，可能知道对文本和其他元素进行旋转、缩放和斜切变换。现在，通过 CSS3 变换在浏览器中也能实现同样的效果了（如图 7–15 所示）。

CSS3 为变换规定了两个属性：transform 和 transform-origin。先说说 transform。

transform 属性能够调用函数，调用不同的变换函数可以实现不同形式的变换，而通过传入的参数值可以控制变换的结果。通过 transform 属性调用变换函数的语法如下：

```
transform:函数名(数值或 x、y 值);
```

以下是 CSS3 规定的变换函数。

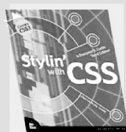
- scale：用于放大或缩小元素（指定大于 1 的值放大元素，小于 1 的值缩小元素），如 transform: scale(1.5)。
- rotate：根据指定的度数旋转元素（正值顺时针旋转，负值逆时针旋转），如 transform: rotate(-30deg)。
- skew：让元素在 x 轴和 y 轴方向倾斜（只指定一个值，y 轴不受影响），如 transform: skew(5deg, 50deg)。

❑ translate：根据指定的距离沿 x 轴和 y 轴平移对象（很像相对定位，因为对象初始占据的空间会保留），如 transform:translate(-50px, 20px)。

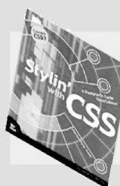
transform-origin 属性设定元素围绕其变换的原点。默认情况下，这个点是元素垂直和水平方向的中心点。因此，如果你旋转元素而未另行指定原点，就会像在元素中心点插进一根大头针一样，然后元素围绕该点旋转。可以使用 transform-origin 属性及位置关键字（left、center、right、top 和 bottom 等）另行设定原点，而使用正、负数字甚至可以把原点设定到元素边界之外。



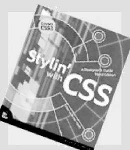
未应用变换



transform:skew(10deg);



transform:skew(-10deg,30deg);



transform:rotate(20deg);



transform:rotate(20deg);
transform-origin:bottom right;



transform:translate
(30px,-30px);

图 7–15　几个变换的例子

现在该为每本书添加弹出层了。以第 6 章那个弹出层示例为基础，还要进行两点改进。首先，让页面右侧图书的弹出层显示在封面图片左侧，以免出现在右侧被浏览器窗口“切掉”；其次，把箭头图形放在弹出层一边，让人感觉它也是弹出层边框的一部分。经验告诉我们，像这种看似不起眼但却很有意思的改进，往往要编写很多代码。

如果你现在看看前面的标记，会发现我已经给包含图书的 article 元素分别添加了 left 和 right 类，以便分别设定出现在封面右侧和左侧的弹出层和箭头。下面就是 CSS 啦！

```
#book_area article aside { /*弹出层共享样式开始*/
    display:none; /*隐藏弹出层*/
    position:absolute; /*相对于包含图片的内部 div*/
    z-index:2;
    width:200px; /*弹出层宽度*/
    background:#fff;
```

```
    padding:10px 2px 5px; /*弹出层内容边距*/
    border:2px solid #f58c21;
    border-radius:10px 0px 10px 0px;
    box-shadow:4px 4px 16px #555;
    color:#555;
    font-family:"Source Sans Pro", helvetica, sans-serif;
    font-size:.8em;
    line-height:1.5em;
    }
#book_area article:hover aside { display:block; }/*鼠标悬停于封面时显示弹出层*/
#book_area article aside li {
    padding:.25em 0 .75em 1em; /*列表项的垂直间距和左边距*/
    list-style-type:none; /*去掉默认的项目符号*/
    line-height:1.2em;
    }
#book_area article aside li a { /*链接文本*/
    text-decoration:none;
    font-size:1.2em;
    color:#616161;
    }
#book_area article aside li a:hover { /*悬停时突显链接*/
    color:#333;
    } /*弹出层共享样式结束*/

#book_area article.left aside {/*左侧两本书*/
    left:84%; top:14px; /* 把弹出层定位在图片右侧*/
    }
#book_area article.right aside {/*右侧两本书*/
    right:84%; top:14px;/*把弹出层定位在图片左侧*/
    }
#book_area article aside:after { /*橙色三角形*/
    content:""; /*需要有内容，这里是一个空字符串*/
    position:absolute; /*相对于弹出层定位*/
    top:33px;
    border:12px solid;
    height:0px; width:0px; /*收缩边框创造三角形*/
    }
#book_area article.left aside:after { /*左侧图书弹出层的三角形定位及颜色*/
    right:100%;
    border-color:transparent #f58c21 transparent transparent;
    }
```

```
#book_area article.right aside:after { /*右侧图书弹出层的三角形定位及颜色*/
    left:100%;
    border-color:transparent transparent transparent #f58c21;
    }
#book_area article aside:before { /*白色三角形*/
    content:""; /*需要有内容，这里是一个空字符串*/
    position:absolute; /*相对于弹出层定位*/
    border:8px solid;
    height:0px; width:0px; /*收缩边框创造三角形*/
    z-index:100; /*保证白色三角形在最前面*/
    top:37px;
    }
#book_area article.left aside:before { /*左侧图书白色三角形的样式、位置和颜色*/
    right:100%;
    border-color:transparent white transparent transparent;
    }
#book_area article.right aside:before {  /*右侧图书白色三角形的样式、位置和颜色*/
    left:100%;
    border-color:transparent transparent transparent white;
    }
```

弹出层是一个绝对定位的元素，它相对于包含每本书的内部 div 定位。在第 6 章那个弹出层的例子中，我们只把红色三角形定位在弹出层的一边（参见图 6-24）。而在这个例子中，我们大幅度地改进了这个效果，弹出层似乎是把边框给向外挤出了一个尖尖的指针，如图 7-16 所示。这个效果是通过在橙色三角形上面叠加另外一个更小的白色（与弹出层背景色相同）的三角形，并将它们垂直的边对齐实现的。这样就用橙色三角形创造了边框的效果。



图 7-16　节省空间的弹出层提供每本书的链接和额外信息

添加白色三角形使用的是::before 伪元素，添加橙色三角形使用的是::after 伪元素，而绝对定位和 z-index 属性则将它们在弹出层边框上精确地对齐。这个结果非常自然，弹出层及其指针浑然一体。

这个例子中容易搞混的地方是记住哪个弹出层在左，哪个弹出层在右。比如，左侧两本书的弹出层位于它们封面的右侧，而这两个弹出层的指针三角形则位于弹出层左侧。右侧两本书的情况恰好相反。

通过这个例子，我们又重温了前面提到的自上而下的方法。前面代码中的注释已经明确告诉你了，首先是所有弹出层共享的样式，包括大小、内边距、边框、颜色和内容等样式。其次是把前两本书的弹出层定位在图片右侧的规则，以及把后两本书的弹出层定位在图片左侧的规则。随后是所有图书弹出层指针三角形共享的样式。再接着是左侧两本书弹出层三角形的样式，以及右侧两本书弹出层三角形的样式。



这种编码原则简称为 DRY（don't repeat yourself），即“不要重复自己”。目标是让每一个效果最终都有“独立且权威的来源”。

尽管一两句话不能解释清楚，但这些代码逻辑上是前后相继的，而且避免了那种每个弹出层都完全重写一遍规则的做法。我想，这个例子很值得大家花点时间深入研究、认真体会。如果能自己从头写出这个例子来，对理解其中的奥妙绝对是有帮助的。这种前后相继的代码组织技巧，能够把为多个元素编码的工作分解成相同的部分和不同的部分。最重要的是，它能为你自己和他人将来的维护工作提供方便。

好了，这个页面就剩最后的页脚啦。

7.5 页脚

页脚非常适合放一些声明之类的信息，比如是谁创建了这个网站，以及一些业务相关的链接，比如免责声明、服务条款、联系信息、隐私政策和版权声明等。下面是页脚的 HTML 标记。

```
<footer>
  <p>A CSS template from <a href="http://www.stylinwithcss.com"><em>Stylin'
        with CSS, Third Edition</em></a> by Charles Wyke-Smith</p>
        <nav>
            <ul>
                <li><a href="#">Privacy Policy</a></li>
                <li><a href="#">Contact Charles</a></li>
            </ul>
        </nav>
</footer>
```

下面是它的 CSS。

```
footer {
    padding:.5em 0 .35em 0; /*内容上下的间距*/
    text-align:center; /*居中内容*/
    border-radius:10px 0px 10px 0px;
    background:#fff;
    box-shadow:0 12px 8px -9px #555;
    }
footer p { /*文本行的样式*/
    font-family:'Source Sans Pro';
    font-weight:400;
    font-size:.85em;
    letter-spacing:-.05em;
    color:#555;
    }
footer p a { /*文本行中的链接*/
    font-family:'Source Sans Pro';
    font-style:italic;
    font-weight:700;
    font-size:1em;
    color:#4eb8ea;
    text-decoration:none;
    }
footer p a:hover {
    color:#777;
    }
footer ul { /*链接列表*/
    display:inline-block; /*收缩包围列表*/
    margin:4px 0 0;
    }
footer li {
    list-style-type:none; /*去掉默认的项目符号*/
    float:left; /*让列表项水平排列*/
    font-family:"Source Sans Pro";
    font-weight:400;
    font-size:.85em;
    }
footer li + li a {
    border-left:1px solid #ccc; /*链接分隔线*/
    }
```

```
footer li a {
    text-decoration:none; /*去掉链接默认的下划线*/
    color:#aaa;
    padding:0 5px; /*链接间距*/
    }
footer a:hover {
    color:#777;
    }
```

A CSS template from *Stylin' with CSS, Third Edition* by Charles Wyke-Smith
Privacy Policy | Contact Charles

图 7-17　包含一个文本元素和一个列表元素的页脚

这一块没有什么新东西，都是你知道的。如图 7-17 所示，内容通过 `text-align:center` 居中对齐，段落及其文本都继承了这一设定，在 `footer` 内居中。这个声明正常情况下不会让链接列表居中，因为列表项由块级元素构成，默认会与容器同宽。不过，我给 `ul` 设定了 `display:inline-block`，让它收缩包围 `li` 元素。这样实际上就相当于为 `ul` 指定了宽度，因此 `text-align:center` 对它也会起作用。记性好的读者应该知道，在前面介绍菜单的时候，我们解释了应用 `display:inline-block` 之后，元素宽度仍然是可变的。换句话说，即使将来再向这个列表中添加链接，或者从中删除链接，列表仍然会居中。最后，再提醒你一下，对列表应用自动外边距（而不对 `footer` 应用 `text-align:center`），同样能让它在页脚内居中。

7.6　小结

整个页面布局到现在已经讲完了，本章也该结束了。相信大家已经会使用第 5 章的结构化布局技术和第 6 章的界面组件快速制作完整的页面了。

下一章，我们还会以本章的页面为例。虽然这个页面在本章中是针对大屏幕显示器创建的，但今天，谁也不能保证浏览你网页的一定是大屏幕设备。因此，在制作网页时必须考虑各种屏幕大小，让页面能够响应设备。相信你一定不情愿维护一堆不同布局大小的页面，然后尝试根据设备屏幕交付其中的某个版本。没错，我们应该让页面检测自己所处的环境，然后加载相应的 CSS，把标记变成适应相应设备的样式。本书最后一章就来教大家创建响应式网站。

8

响应式设计

今天的网页布局必须能根据它自己所处的不同环境作出响应。大屏幕上的最佳体验和手机中的最佳体验有着天壤之别。在大屏幕上，可能使用多栏布局效果很不错，但多栏布局到了手机上，每一栏都会窄得没法看。此时，所有内容“鱼贯而行”，即用一个栏来组织成了唯一可行的方案。这样，用户使用简单的扫屏手势，就可以轻松自如地滚动屏幕来阅读了。

实际上，使用一项叫媒体查询的 CSS 功能，很容易检测出用户设备的屏幕大小。然后，据以提供替代或额外的 CSS，可针对相应屏幕实现更加优化的体验。使用这种方式创建对设备有感知力的网站，被称为响应式设计。

本章仍然以第 7 章的那个为桌面浏览器设计的页面为基础，向大家依次展示在屏幕越来越小的设备上优化这个页面的全过程。



虽然我们是要修改浏览器版以适应小屏幕，但实际上真正的设计原则是“移动先行”。推荐大家读一读 Luke Wroblewski 的文章：http://www.lukew.com/ff/entry. asp?933，还有 A Book Apart 出版的他的一本书：*Mobile First*。

8.1 小设备上的大布局

我们在这里以 iPad 和 iPhone 作为目标设备，但背后的概念同样适用于其他平板电脑和智能手机。首先，让我们在小屏幕上看一看第 7 章的页面（固定宽度 980 像素布局）。

iPad 屏幕尺寸是 1024 × 768 像素，由于页面布局宽度为 980 像素，因而在横向（水平）的时候恰好能容纳布局（见图 8-1）。当把 iPad 竖过来（垂直）以后，页面布局就放不开了。



iPad 和 iPhone 屏幕及其界面组件的大小，详见这里：http://upstageapp.com/resources。



图 8-1　在 iPad 横屏的时候，第 7 章的网页看起来很舒服，而且能填满屏幕；在 iPad 竖屏的时候，页面看起来小了一号，也没有填满屏幕

好了，再看看通过 iPhone 查看这个页面会怎么样，见图 8-2。



图 8-2　在 iPhone 横屏的时候，页面能填满屏幕，但已经小得几乎看不清文字了；在竖屏的时候，页面就更小了

如你所见，iPad 和 iPhone 会自动缩小页面以填满屏幕，但页面布局在 iPhone 的小屏幕上根本没法看。特别是文字太小了。用户必须通过扩展手势来放大页面，以便看清文字。只是这样一来，每次都只能看到屏幕中的一小块地方。在竖屏的时候，也就是用户自然而然地把握手机的情况下，页面还会更小。

很明显，一种布局不能适应多种屏幕尺寸。我们需要一种能够检测屏幕大小的方法，然后相应地修改布局。简言之，就是需要让页面能够自己响应屏幕变化。下面我们就来看一看实现这一点需要哪些技术。首先就是媒体查询。

响应式设计的要素

响应式设计包含三个重要的方面。

- ❑ 媒体查询：是一种 CSS 语法，可以根据浏览器的特性，一般是屏幕或浏览器容器宽度提供 CSS 规则；
- ❑ 流式布局：是使用 em 或百分比等相对单位设定页面总体宽度，让布局能够随屏幕大小而缩放；
- ❑ 弹性图片：是使用相对单位确保图片再大也不会超过其容器。

这几个方面最早是由 Ethan Marcotte 提出来的，发表在 2010 年 5 月份的 *A List Apart* 杂志上，地址是：http://www.alistapart.com/articles/responsive-web-design。

8.2　媒体查询

媒体查询是 CSS 代码的容器，其中的 CSS 只在某些条件（比如，当前页面要被打印或者要显示在某种类型或尺寸的屏幕上）具备时才会应用。媒体查询可以用两种方式来写：`@media` 规则和`<link>`标签的 `media` 属性。



安卓设备的屏幕大小千差万别，下面这个网页列出了针对 iOS、安卓和 Windows 设备的所有媒体查询：http://pugetworks.com/blog/2011/04/css-media-queries-for-targeting-different-mobile-devices/。

8.2.1　@media规则

第一种方式是@media 规则，可以在样式表或<style>标签的 CSS 中包含媒体查询，比如：

```
@media print {
    nav {
        display:none;
    }
}
```

这条规则声明：如果当前页面要打印，那么就不显示 nav 元素。

大家注意，这里是把 CSS 规则嵌套在了一个@media 规则中，乍一看似乎有点不太习惯。尽管可以把 CSS 规则嵌套在媒体查询里，但媒体查询本身却不能互相嵌套。下面再看一个假设的示例（倒是跟图 8-1 和图 8-2 中的例子关系更密切），其中涉及最大屏幕宽度。

```
/*只在屏幕宽度不大于 568 像素时应用*/
@media screen and (max-width:568px) {
    .column {float:none; width:96%; margin:0 auto;}
}
```

对这个例子而言，如果页面是通过屏幕显示，而且该屏幕宽度不超过 568 像素，那么 CSS 就会取消带.column 类的元素的浮动，让布局区块上下堆叠，且让该元素宽度为屏幕的 96%，同时在屏幕上居中。

iPhone 4 的屏幕分辨率为 320 × 480，而 iPhone 5 的屏幕分辨率则为 320 × 568（至少对浏览器和媒体查询来说是这么大，因为像素是翻倍使用的——物理像素在两个方向上都是这里值的两倍）。把 max-width 设定为较大的 iPhone 5 屏幕的最大宽度，可以保证布局各栏在所有 iPhone 中都不再浮动，而是上下堆叠。



想知道怎么在苹果视网膜（retina）屏和其他高分辨率屏上以恰当的分辨率显示图片？可以参考这篇文章：http://coding.smashingmagazine.com/2012/08/20/towards-retina-web/。

简单地理解，就是相应设备必须是一块最大屏幕宽度不超过 568 像素的屏幕（比如浏览器或智能手机）。因此，这条规则就不会应用给 1024 × 768 像素的 iPad。

下面我们再看一看如何在`<link>`标签的 `media` 属性里写媒体查询。

谈谈媒体查询

媒体类型　最常用的媒体类型如下所示。

- ❑ `all`：匹配所有设备；
- ❑ `handled`：匹配手持设备（小屏幕、单色、带宽有限）；
- ❑ `print`：匹配分页媒体或打印预览模式下的屏幕；
- ❑ `screen`：匹配彩色计算机屏幕；
- ❑ 其他媒体类型还有 `braille`（盲文点字触觉反馈设备）、`embossed`（盲文分页打印机）、`projection`（投影仪）、`speech`（语音合成器）、`tty`（电话机屏幕等固定宽度字符栅格设备）和 `tv`（电视机）。

要想详细了解这些媒体类型，请参考 CSS 2.1 标准：http://www.w3.org/TR/CSS2/media.html。

当然，任意时刻浏览器窗口中只能使用一种媒体类型。媒体类型从 IE6 开始就得到支持了，但媒体特性到 IE9 以上才得到支持。一般来说这并不是问题，因为我们使用媒体特性多数情况下都是为了检测平板电脑或智能手机等现代设备。

媒体特性　媒体特性也就是媒体某一方面的特征，一般带有 `min-` 或 `max-` 前缀。最常用的媒体特性如下。

`min-device-width` 和 `max-device-width`：匹配设备屏幕的尺寸；

`min-width` 和 `max-width`：匹配视口的宽度，例如浏览器窗口宽度；

`orientation`（值为 `portrait` 和 `landscape`）：匹配设备是横屏还是竖屏。

如果想通过媒体查询来根据用户对浏览器窗口的缩放重新调整布局，应该使用 `min-width` 和 `max-width`。要了解所有媒体特性，请参考 CSS3 标准：http://www.w3.org/TR/css3-mediaqueries/#media1。

可以使用逻辑运算符 `and`、`not`、`or` 及关键字 `all`、`only` 组合媒体类型和媒体特性。

其中，only 关键字可以用来对不支持媒体查询的浏览器隐藏样式表。关于媒体查询中可以使用的逻辑运算符的详细信息，请参考这里：https://developer.mozilla. org/en-US/docs/CSS/Media_queries#Operator_precedence。

这里有一篇关于媒体查询的扫盲文章，非常不错：http://www.javascriptkit.com/dhtmltutors/cssmediaqueries.shtml。至于要在 IE8 及以下版本的 IE 浏览器中使用媒体查询，可以使用腻子脚本 Respond.js（参见本书附录末尾的“腻子脚本”）。

8.2.2 <link>标签的media属性

如果要通过媒体查询应用的 CSS 规则非常多，那么就可以考虑使用<link>标签的 media 属性设定条件，有选择地加载独立的样式表。你不会不知道吧，我们前面一直都是在使用<link>标签向 HTML 中链接样式表的。嗯，这个你应该知道。不过，你未必知道的是，通过在<link>标签的 media 属性中指定条件，可以有选择地加载样式表。

下面这个例子在前面介绍@media 规则时展示过，但这次我们使用的是<link>标签的 media 属性。

```
<link type="text/css" media="print" href="css/print_styles.css" />
<link type="text/css" media="screen and (max-width:568px)"
href="css/iphone_styles.css" />
```

结果都一样，即这里的 CSS 样式表会根据 media 属性中的指定的条件应用。而且，查看页面的如果是大显示器或 iPad，则浏览器根本不会加载上面例子中的第二个样式表。第二个样式表只会被查看该页面的智能手机加载。那怎么使用媒体查询最直观呢？最直观的方式莫过于根据不同的断点来编写媒体查询。



要适配不同的设备，推荐 Andy Clarke 的 320 and Up 样板包：http://stuffandnonsense.co.uk/projects/320andup/。

8.2.3 断点

断点（breakpoint）在这里指的是媒体查询起作用的屏幕宽度，其写法类似如下形式。

```
@media screen and (max-width:640px) { /*CSS 规则*/ }
```

在这里，断点是 640 像素宽。如果有设备的屏幕宽度等于或小于断点设定的宽度，那么后面的 CSS 就会起作用。

可能有人会使用断点去匹配特定设备的屏幕宽度，但我认为最重要的，还是在屏幕变小的情况下，通过简化布局来确保可用性。换句话说，不要简单地用断点去匹配设备宽度，而是可以从慢慢地缩小浏览器窗口开始，在发现当前布局不合适的时候再确定断点，然后编写新样式。不要针对某款具体的设备，而要为宽度在某个范围内的屏幕提供替代的布局，该布局对于范围内的设备都应该适用。



我在发现一个可以作为断点的布局宽度时，通过查看我的电脑上安装的 Ambrosia Software 的截屏软件 SnapzPro X 所显示的大小，就可以知道布局的像素宽度。

通过缩小浏览器窗口，能够直观地感受到当前布局在小屏幕上的效果。不过，毋庸置疑，必须还要在小屏幕设备上进行测试。平板电脑与小显示器尺寸接近，因此 1000 像素的断点（为屏幕小于等于 1000 像素的设备提供样式）适合平板电脑。在这样做之前，我们先讲一讲怎么取消 iPhone 和 iPad 中“缩小适应”的默认行为。

8.2.4　用<meta>标签设定视口

从图 8-1 和图 8-2 可以看出，iPad 和 iPhone 会把适合大屏幕的网页缩小，以便在它们较小的屏幕上能看到网页的全貌。这是一个通用技巧，但对于手机——特别是 iPhone 来说，由于文字实在太小了，为了看清楚网页内容，肯定得用扩展手势放大页面，然后再来来回回地扫屏。如果你想让自己的页面布局适合这些小屏幕，首先就要覆盖这种自动缩小的设定。方法是在页面的`<head>`标签里添加一个`<meta>`标签：

```
<meta name="viewport" content="width=device-width; maximumscale=1.0" />
```

这个`<meta>`标签告诉浏览器按照屏幕宽度来显示网页，不要缩小网页。虽然这样可以让布局以实际宽度显示，但在 iOS 设备（如 iPad 和 iPhone）中却会引发一个已知的 bug。关于这个 bug 及如何解决，本章后面再交待。



关于`<meta>`标签，其实还有很多内容值得深究，详细情况请参考这里吧：http://developer.android.com/guide/webapps/targeting.html。

8.3　针对平板优化布局

第 7 章的网页（或布局）目前是固定的 980 像素宽，而且在比 980 像素更宽的浏览器窗口内会居中。随着浏览器窗口变窄，比如在平板电脑或手机上浏览这个页面时，布局的右边会被浏览器窗口切掉。为此，我想第一个断点应该设定在布局的固定宽度之上，即 1000 像素。如果浏览器窗口变得比这个断点还要窄，那么我希望布局能够变成流动的，以便它能够自己缩小宽度来适合浏览器窗口。回忆一下，这个网页布局中的所有结构化元素，使用的都是 auto 或者百分比宽度值，因此从固定到流动的转换很简单，只要把固定宽度的外包装元素设为百分比宽度就行了。



百分比宽度体现了响应式设计的第二个要素——流动布局。

```
/*98%可以保证两边有一点边距*/
#wrapper {width:98%;}
```

不过，布局变成流动之后会出现几个问题。



图 8-3　在 iPad 竖屏状态下，布局中的区块交叠到了一起

如图 8-3 所示，随着布局宽度变窄——这里是在 iPad 竖屏，即 768 像素宽度的情况下，导航菜单已经压在了其左侧的页面标题上面。而中间的登录表单和博文链接也都盖住了部分博文内容。显然，菜单的定位方式及页眉下方的两栏布局不适合这么窄的屏幕。

那好，这就是我们在 1000 像素断点中需要解决的问题。还是先来看一看断点中的 CSS 规则，之后再听我解释。

```
@media only screen and (max-width:1000px) { /*1000像素的断点*/
    body {
        margin:0 8px 20px; /*添加右外边距，以防滚动条碍事儿*/
    }
    #wrapper {width:98%;} /*布局由固定变成流动*/
    header {
        height:100px; /*增加页眉高度，为重新定位菜单留出空间*/
        padding:1px 0 0 0; /*防止导航菜单的上外边距叠加*/
    }
    header nav.menu {
        margin-top:65px; /*把菜单移动到页面标题和搜索框下方 */
    }
    section#feature_area {padding-bottom:0;} /*用不着了*/
    section#feature_area article { /*让博文摘要部分与布局同宽*/
        float:none; /*不需要浮动了*/
        width:auto;/*自动填满布局*/
    }
    section#feature_area aside { /*原来的右栏同样与布局同宽*/
        float:none;
        width:auto;
    }
    section#feature_area aside form {
        float:left; /*浮动到左侧*/
        margin:15px 0 0 0; /*与nav的上外边距一致*/
    }
    section#feature_area aside nav {
        width:17em; /*缩小博文链接区的宽度*/
    }
} /*1000像素断点结束*/
```

图 8-4 展示了应用以上断点规则后，通过 iPad 竖屏模式浏览页面会看到不一样的效果。iPad 横屏时的宽度大于 1000 像素，因此不会应用以上规则，仍然是图 8-1 中的

样子，而在竖屏时由于宽度小于 1000 像素，所以就会得到新的流动布局。

如图 8-4 所示，通过媒体查询应用新规则之后，页眉高度和菜单上外边距都增加了。这些变化让菜单移到了页眉底部新增的空间，它现在位于标题和搜索框之下了。



图 8-4　当布局窗口变得小于 1000 像素之后，就会应用媒体查询中的 CSS 规则。菜单移到下方，而原来右栏的元素重新定位到了左栏下方

feature_area 也不一样了，它的两个子元素——包含博文摘要的 article 和包含登录表单及博文链接的 aside 被取消了浮动并将宽度重设为 auto。它们由左右并列变成了上下堆叠，并分别填满布局宽度。在 aside 内部，表单向左浮动，因而 form 和 nav 并列在一起，共同位于 article 元素下方（nav 本来就是向右浮动的）。另外，为了在布局宽度缩小到下一个断点（640 像素）之前，不让 nav 元素碰到表单的右边，我们还把 nav 元素稍微设定得窄了一些。（下一个断点的情况稍后介绍。）

此时，还不需要对 book_area 写什么新样式，因为一开始我们就给每本书的容器都设定了 25%的宽度，并没有设定固定的宽度。所以，这些图书封面及各自相应的文字只会随着布局变窄而靠得更近。

在 iPad 或其他窗口宽度小于 1000 像素的非移动浏览器中，这种布局会给人更好的阅读体验。好啦，接下来再看一看如果布局宽度进一步变窄又会如何。

8.4 针对智能手机优化布局

可是，这种改进的效果只能在一定范围内保持。如果布局宽度到了 640 像素以下（达到智能手机屏幕的宽度），那么第二行的 form 和 nav 接触，而它们下面的图书封面也会重叠。

我们需要在这个宽度上增加一个断点。

```
@media only screen and (max-width:640px) {
    header {height:100px;}
    header nav.menu {width:94%; font-size:.65em; } /*水平间距更多了*/
    section#book_area article { /*博文摘要*/
        width:auto;
        float:none;
        margin:0; padding:0; /*每本书的容器都与布局同宽*/
    }
    section#feature_area aside form,
    section#feature_area aside nav { /*博文链接*/
        margin:10px auto; /*添加上、下外边距*/
        float:none;
    }
    /*图片容器与布局同宽*/
    section#book_area article .inner {width:98%; margin:0 0 0 5px; }
    #book_area  .inner h3 { /*取消文本旋转效果*/
        -webkit-transform:none;
        -moz-transform:none;
        -moz-transform-origin:none;
        -ms-transform-origin:none;
        transform:none;
        position:static;
    }
    #book_area article img { /*相对设备宽度确定图片宽度 */
        width:40%;
    }
    section#book_area {background:#fff; padding: 0 10px 10px; margin:0 0 10px;}
```

```
    #book_area article aside { /*在图片旁边显示弹出层*/
        display:block;
        position:static;
        float:right;
        margin:0; padding:0 0 20px 0;
        font-size:.8em;
        border:none;
        width:55%;
        box-shadow:none;
    }
    section#book_area article aside::before, /*隐藏弹出层的三角*/
    section#book_area article aside::after {
        display:none;
    }
}
```

640 像素的断点比 1000 像素的断点作出了更多改变。首先，此前并排的 form 和 nav 元素被取消浮动，变成与布局同宽，因而上下堆叠起来。其次，包含图书封面的容器，也由原来并列的一行，变成了现在的上下堆叠。此时，图书封面旁边的标题文字也没有必要再旋转了，它们在 HTML 标记里原本就在图片前面，只要去掉对文本的旋转声明，这些标题就会按照各自默认的 static 定位方式，自然地领衔每本书。弹出层也变得不再必要，因此要取消隐藏它们的声明及相关样式，把它们定位到每本书的旁边。以上这些改变得到了一个单栏、序列化的布局，非常适合智能手机。目前来看，页面中所有主要的元素都与布局同宽，而且相互堆叠在一起。

如图 8-5 和图 8-6 所示，我们还把图片宽度设定为 40%，确保它们不可能比布局更宽。这个例子中使用的图片比较小，假如你使用的图片在小屏幕设备上比它们的容器还要宽，那么就需要在样式表中包含下面这条规则：

```
img {max-width:100%;}
```

这条规则可以确保这些大图永远不会比它们的容器更宽，同时也不会影响它们缩小。这条规则体现了响应式设计的第三个要素：弹性图片。

布局中所有堆叠元素的宽度都是用相对于浏览器窗口的百分比设定的，因此当设备在横屏和竖屏间切换时，布局元素能够从容地重新排放。



图 8–5 单栏横屏布局在 iPhone（和 iPad）中效果不错



图 8–6 竖屏时只有页眉显得太大了

针对竖屏进一步优化

如图 8-6 所示，iPhone 竖屏时的页眉存在两个问题。一是菜单折行，二是搜索框压住了标题。这两个问题都很容易解决，为此我们要专门为 iPhone 竖屏再添加一个断点。虽然可以在媒体查询中使用 portrait 关键字，但我不想让 iPad 竖屏也应用这些样式。所以，这个断点依旧使用像素宽度，即将 max-width 设定为 320 像素，也就是 iPhone 竖屏时的屏幕宽度。

```
@media only screen and (max-width:320px) { /*iPhone 竖屏*/
    header {height:90px;} /*缩小页眉高度*/
    header section#title h1 {font-size:1.25em;} /*文本再小一点*/
    header section#title h2 {font-size:.75em;} /*文本再小一点*/
    header form.search {top:6px; right:2px;} /*搜索框上移*/
    /*按比例缩小，并上移菜单*/
    header nav.menu {font-size:.55em; margin-top:55px;}
    nav.menu ul li a {
        padding:5px 4px; /*增大链接，方便点击*/
        margin:0;
    }
}
```



图 8-7　在 iPhone 竖屏的 320 像素宽度下，布局也非常漂亮了

如图 8-7 所示，以上 CSS 减少了页眉高度，把搜索框移到了右上角。通过设定较小的相对字体大小，菜单也更小了，因为菜单大小完全由链接的文本大小和围绕文本的内边距决定。既然链接文本和菜单元素变小了，那就可以把页眉的垂直高度再减少一点，腾出一点宝贵的空间。

现在，上至宽屏显示器，下至竖屏智能手机，都能获得最佳用户体验了。不过，要完成我们的响应式设计，还有一些细节问题要解决。

8.5　最后两个问题

还有最后两个问题需要解决。一个是有详细文档说明的 iOS（苹果移动操作系统）设备上的重绘和缩放的 bug，另一个是让下拉菜单也支持触摸。

8.5.1　移动Safari中的缩放bug

Safari Mobile（iPhone 浏览器）中有一个 bug，在设备从竖屏旋转到横屏时会导致缩放和重绘问题，如图 8-8 所示。

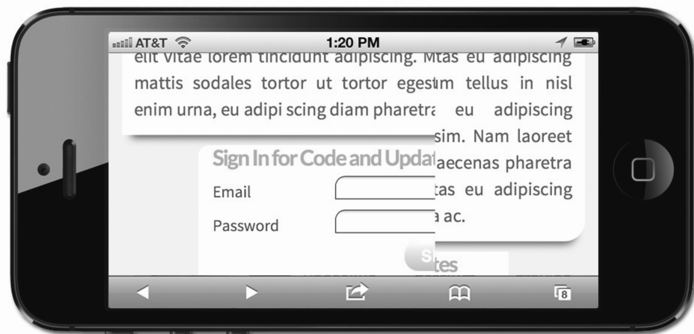


图 8-8　在从竖屏旋转到横屏时，WebKit 中的 bug 会导致屏幕重绘

有一个 JavaScript 脚本可以解决这个重绘问题，我已经把它包含在完成后的 HTML 示例文件中了。要了解关于这个 bug 的更多信息，请参考这里：http://webdesignerwall.com/tutorials/iphone-safari-viewport-scaling-bug。访问这里下载脚本：https://gist.github.com/901295。

8.5.2　让下拉菜单支持触摸

最后，还有一个问题要解决。那就是让下拉菜单支持触摸操作。还记得吗，上一章，我们组合使用了 opacity 和 visibility 属性，实现了下拉菜单淡入淡出的效果。当时使用的 CSS 如下。

```
nav.menu li ul { /*隐藏下拉菜单*/
    opacity:0; visibility:hidden;
    -webkit-transition:1s all; /*同时过渡 opacity 和 visibility 属性*/
    -moz-transition:1s all;
    transition:1s all;
}
/*鼠标悬停时显示下拉菜单*/
nav.menu li:hover ul {opacity:1; visibility:visible;}
```

问题在于，支持触摸的设备会跳过:hover 规则中对 visibility 属性的过渡。似乎也不算太意外，毕竟这个属性只是简单地切换了一个布尔值的状态，从而实现显示和隐藏。但这也说明 visibility 属性不是过渡动画的理想对象。除非我去掉 visibility 属性，否则菜单在触摸屏上没法用。可是如果真这样做，在非触摸屏上只要鼠标经过菜单下方，就会导致下拉菜单显示出来。因为下拉菜单虽然处于完全透明状态，但它对鼠标还是“可见的”。解决方案是使用 Modernizr 检测设备是否支持触摸，如果支持再去掉对 visibility 属性的过渡。如果设备支持触摸，Modernizr 会给根元素 html 添加一个 touch 类，我们就可以针对触摸设备单写一条规则：



关于使用 Modernizr，这里有一篇非常精彩的文章：http://webdesignernotebook.com/css/how-to-use-modernizr/。

```
/*Modernizr 检测到触屏，再去掉妨碍菜单过渡的 visibility 属性*/
.touch nav.menu li ul {
    -webkit-transition:1s opacity;
    -moz-transition:1s opacity;
    transition:1s opacity;
}
```

这条规则对（不支持触摸的）非移动设备是不适用的。在触摸设备上，有了它用户在触摸相关菜单项时，下拉菜单就会显示出来。不过，当用户再触摸别的地方时，

菜单会立即消失，而不是淡出屏幕。实际上，淡出效果的唯一好处，就是当用户鼠标意外离开时，再移回去还能把菜单“召回”。因此，在不使用鼠标的触摸设备上，这种“突然消失”的效果还是可以接受的。

请大家注意，这个菜单只有在触摸屏幕其他地方的时候才会消失。因为触摸其他地方会触发收缩 bug 的代码（上一节介绍过），强迫 JavaScript 运行，这样就足以让菜单意识到当前已经不再是悬停状态而关闭了。另一个实现这种“触摸其他地方关闭”效果的方法是执行某个 JavaScript 函数。比如，下面就是执行 jQuery 的 noop 函数（英文 noop 的意思是“无操作”）的例子。虽然这个函数什么也不做，但只要通过触摸屏幕其他地方来调用它，就可以关闭菜单了。

```
(function(){ $(window).on('touchstart',$.noop); })();
```

当然，这完全是一种对付的手段。不过，JavaScript 专家 Isaac Shapira（他告诉我的这个技巧）跟我说：“这是一种优雅的对付。”假如你不需要修复缩放 bug，那么只要加上这行代码就可以关闭菜单了。



要是你连 jQuery 都不想用，可以只在<body>标签上添加 ontouchstart=" "。这样也会在用户触摸屏幕其他地方时触发 JavaScript 运行，从而关闭菜单。

博客圈里一直都在热烈地讨论怎么让触摸设备支持下拉菜单。我给大家提供这么一个链接，大家有空可以看一看，里面有一个试验性的例子：http://css-tricks.com/convert-menu-to-dropdown。

我是在写这本书的时候，才最终找到这个让下拉菜单既能在鼠标设备又能在触摸设备上使用的方法。这个方法并不完美，我还会继续完善它。因此，请大家有空就到我的博客上瞧瞧，没准儿我又找到了更可靠的办法，或者又发现了其他问题。不过，当前这个方法在 iPad 和 iPhone，以及有限几款 Android 设备上都没问题，我都测试过了。

好吧，就这样了。我们这个几乎能在任何设备上恰当展示的网页已经可以告一段落了。随着移动设备在人们工作生活中扮演的角色越来越重要（越来越多的互联网流量来自移动设备），开发响应式网站也只会越来越重要。

8.6　小结

这个练习结束了本章，同时也结束了这本书。本章，我们看到了如何在响应式设计中运用媒体查询、流动布局和弹性图片，让同样的网页能够适应不同屏幕大小的设备。对于实现响应式设计中的一系列问题，我没有完美的方案，而且任何人也不会有。因为响应式设计是当前最新的 Web 设计思想，而技术、浏览器和硬件都在迅速发展变化之中。

CSS，特别是 CSS3 本身，就有许许多多可能的发展方向，这些方向的进度又快慢不一。因此，学习使用 CSS 万万等不得。你要做的就是立马一头扎进去，只要知道什么能用，什么暂时还不能用，然后为将来的变化提前作好准备就行啦。很明显，随着 HTML5 和 CSS3 日益普及并支持更多类似原生应用的能力，Web 必将发展到一个崭新的阶段。我衷心希望这本小书能为你打开一扇门，让你看到各种可能性，激发你产生伟大的想法，并能够把它们变成现实。

附录

技术提示

编写CSS

CSS 是 HTML 的排版语言。本书几乎所有例子都是从 HTML 代码块开始的，然后才是 CSS 代码。HTML 代码类似如下所示。

```
<p>A HTML paragraph element</p>
```

浏览器忽略 HTML 代码中的空格符、回车符和制表符，但保留文本之间的空白——多个空白符只保留一个。举个例子，下面这几行全都是等价的 HTML 代码，它们的结果都是显示：An HTML paragraph element。

```
<p>An HTML paragraph element</p>
<p>A HTML paragraph element</p>
<p>
An HTML paragraph element
</p>
```

CSS 代码类似如下所示。

```
p {color:red;}
```

浏览器也会忽略 CSS 中的空格符、制表符和回车符，因此下面这几种写法都是等价的：

```
p {color:red;font-size:20px;line-height:1.2;}
p {color: red; font-size: 20px; line-height: 1.2; }
p {
    color: red;
    font-size: 20px;
    line-height: 1.2;
}
```

前两个例子展示了写在一行中的包含多个声明的 CSS 规则（第二个例子只不过多了几个空格而已）。第三个例子展示了每个声明各占一行的写法。但这三个例子的效果完全相同。在编写 CSS 时，我会混用这些写法（多个声明一行，一个声明一行都有）。另外，虽然一条规则中的声明可以按照任何顺序写出来，但我背后还是有一个优先级顺序：

1. display 及相关声明；
2. position 及相关的声明；
3. margin、padding 和 border 及相关声明；
4. 字体/文本相关声明；
5. 装饰相关声明。

比如：

```
.demo {
    display:block; position:absolute;
    height:100px; width:300px; left:10px; top:10px;
    margin:0 5px; padding:10px;
    font-size:10px; line-height:1.2;
    background-color:#eee; border:1px solid; border-radius:6px;
}
```

这个顺序首先考虑了我认为对元素最重要的信息，即它们怎么在页面上定位，定位到哪里。随后是不那么重要的信息，包括元素的视觉装饰。有时候，我也会根据需要改变这个顺序。比如，要是 margin 声明在相应规则里最重要，就把该声明写在最前头。我认为把相关性强的几条声明都写在一行是一个不错的考虑，因为把所有声明都写在一行，会导致 CSS 代码难以理解，而每个声明各占一行又会让样式表过长，有时候需要来回滚动着看。当然，这完全是个人喜好问题，“仁者见仁，智者见智”。在本书里面，上述几种写法同时存在，有时候是为了适应空格问题，有时候是为了便于添加注释。

不过，无论如何，我都向大家强烈推荐一种组织 CSS 的方法，那就是在样式表中按照接受样式的 HTML 标记出现的先后顺序，依次列出相应的 CSS 规则。千万不能把新样式全都扔到样式表最后去。你可以找一个比较长的样式表，比如本书第 6 章表单的样式表，会发现其中的 CSS 规则与表单的标记顺序完全一致。CSS 样式表有时候一写就会非常长，如果不这样按顺序罗列，将来要想找到为某个元素应用样式的某条规则会非常麻烦。

测试代码

边写代码边调试是不可避免的。什么是调试？调试就是找出你的代码没有得到预期结果的原因。我喜欢一句话，“调试是对假定的系统性侵害”。换句话说，调试过程就是去发现你认为代码该做什么与代码实际上做了什么之间差别的过程。

为此，确切知道某个 HTML 元素被应用了哪些 CSS 规则至关重要。由于 CSS 规则会层叠，很多 CSS 规则都可能设定同一个 CSS 属性，但最终这个属性只能取得一个值。比如，对 `body` 元素设定的字体样式将被页面上的所有文本元素继承。如果有规则为某个文本元素设定了另一种字体样式，则该样式就会覆盖从 `body` 继承的样式。一般来说，找到哪个样式胜出并非易事，因此本节就教给大家一种方法，让你能亲眼看到某个元素都被应用了哪些样式，而哪些样式又是最终起作用的。下面这个例子是从我的电子书 *Visual Stylin' with CSS3* 中找的一个页面。

如图 A 所示，在浏览器（这里是 Safari）中右键单击一个元素，然后从上下文菜单中选择 Inspect Element，就会打开 Web Inspector。Web Inspector 同时会显示 HTML 元素（左下面板）和该元素接受的 CSS 规则（右下面板），如图 B 所示。

这里的 CSS 清单显示，虽然为 `p` 元素设定了 Open Sans 字体，但这个字体样式被上面那条更有针对性的 `.basic_borders_large.demo8 p` 规则覆盖了，看到那条删除线没？因此，其中的文字会用 Niconne 字体显示。在 HTML 面板中指定一个元素，与之相关的样式就会出现在 CSS 面板。

这对我们知道某个元素到底都接受了哪些样式确实太有用了。假如你修改了一个样式，但元素并没有受影响，那你就该打开 Web Inspector 看一看，到底是你修改的规则还是其他规则在实际影响该元素。看完了，单击左上角的关闭（x）按钮，就可以退出 Web Inspector。



图 A（右上）　右击元素并选择 Inspect Element（审查元素），打开 Web Inspector（或在安装了 Firebug 的 Firefox 中打开“查看器”）

图 B（下）　选中元素的 Web Inspector 界面



我个人还是推荐 Firebug，它是 Firefox 的一个扩展。通过 Firebug 不仅能查看 DOM 结构，还能调试 JavaScript。要安装 Firebug，可以在 Firefox 的“Firefox”菜单中选择“Web 开发者”>“获取更多工具”，然后在“附加组件”页面中搜索“Firebug”。找到后按照简单的说明安装即可。

支持旧版本浏览器

HTML5 和 CSS3 给我们提供了那么多容易实现的功能，那自然应该赶紧使用而不管旧版本浏览器是否支持这些新功能。可是，现实是残酷的！在旧版本浏览器中，有些 CSS3 功能可能只是没有效果而已，这倒简单；但有时候，某些 CSS3 功能可能导致不支持它们的浏览器出现显示问题。比如 `display:table` 和 `boxsizing` 属性吧，我们在第 5 章介绍过，用它们实现多栏是非常方便的，但它们在 IE6 和 IE7 中会导致布局一片混乱。因此，在旧版本浏览器中测试一番，然后为相应浏览器提供后备代码是极有必要的。

事实上，直到不久前，浏览器嗅探（browser sniffing）都是检测浏览器的一种流行方式。所谓浏览器嗅探，就是通过 JavaScript 检查浏览器的用户代理字符串中包含的浏览器名字，然后再为之提供能够弥补不足的代码。然而，我们实际上关心的并非浏览器，而是浏览器到底支持什么功能。这就是为什么现在大家使用的方法都不再关心浏览器，而是直接检测功能的原因。检测到功能缺陷，再有针对性地提供后备代码或者腻子脚本，就可以弥补相应能力的不足。

后备代码

后备代码就是给不支持 CSS3 功能的旧版本浏览器提供的一段替代性代码。

最简单的后备就是没有后备，而且很多时候，啥也不准备都没有问题。比如，你用了 CSS3 的圆角功能，IE6 和 IE7 不会显示圆角，这些浏览器的用户看到的还是方角。这个结果好像也没什么大不了的，那些用户也许根本不知道你的网页中有圆角效果，而有没有圆角似乎对用户获取信息也没那么重要。不过，在另外一些情况下，为浏览器提供后备 CSS 代码则是必需的。

举个简单的例子吧，IE9 之前的浏览器都不支持多背景，因此后备代码就是在多背景声明之前简单地再加一条单背景声明，比如：

```
.someElement {background-image:url(images/basic_image.jpg);}
.someElement {background-image:
    url(images/cool_image1.jpg),
    url(images/cool_image2.jpg),
    url(images/cool_image3.jpg);
}
```

所有浏览器都能理解第一条规则，但只有支持多背景的浏览器才会采用第二条规则。如果某浏览器无法解析某条 CSS 规则，可能是因为它不支持其中的 CSS 属性或者声明中包含错误，那么它就会跳过该规则，接着读取下一条规则。因此，IE8 及更早版本的 IE 浏览器会忽略第二条规则，而只呈现 basic_image.jpg。

条件注释

如果你真想单独为 IE 浏览器做点什么，可以使用如下所示的条件注释来添加后备代码：

```
<!--[if lte IE 8]> <!-- IE 条件注释 -->
<link src="ie_only.css" rel="stylesheet" />
<![endif]-->
```

这种特殊格式的 HTML 注释会被非 IE 浏览器忽略，只有 IE 浏览器才会执行其中的代码。就这个例子而言，我们是在为 IE8 及更低版本的 IE 加载额外的样式。加载条件可以使用 `lte`（less than or equal to，小于等于）、`lt`（less than，小于）、`gte`（greater than or equal to，大于等于）、`gt`（greater than，大于），甚至只写一个浏览器版本号如 `IE 6`。以此为不同版本的 IE 提供后备代码。

有时候，浏览器可能根本不支持你需要的功能。此时，仅提供后备 CSS 代码还不够，还需要使用腻子脚本。

腻子脚本

腻子脚本（polyfill）指的是一段 JavaScript 代码，能够赋予浏览器未曾有过的功能。目前，几乎所有 CSS3 和 HTML5 功能都有各自对应的腻子脚本，包括视频回放和阴影，从而让那些老得走不动道儿的“老家伙”们也能闪现青春的光芒。



Paul Irish 维护了一个完整的腻子脚本列表，地址为：https://github.com/Modernizr/Modernizr/wiki/HTML5-Cross-Browser-Polyfills。

要在页面中添加一段腻子脚本，首先要下载并将其保存在网站的一个文件夹中。我为此创建了一个 helpers 文件夹。然后，在页面`<head>`标签中添加一个`<script>`标签把它加载进来。

```
<script type="text/javascript" src="helpers/selectivizr.js">
</script>
```

怎么确定是否需要某个腻子脚本呢？推荐大家使用 Modernizr。Modernizr（http://modernizr.com）是一个 JavaScript 文件，能够帮你检测用户浏览器对 HTML5 和 CSS3 功能的支持情况，然后为顶级的`<html>`标签添加一组类，标明浏览器支持什么功能。另外，它还会设定一个 JavaScript 对象 `modernizr` 的属性，以便你通过 JavaScript 来测试这些功能。Modernizr 添加的类主要是为 CSS 提供便利。

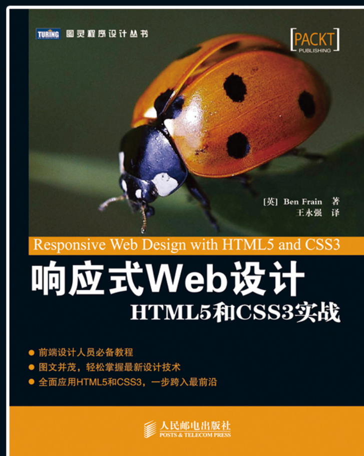
以下给出一些有用的腻子脚本，供大家参考选用。

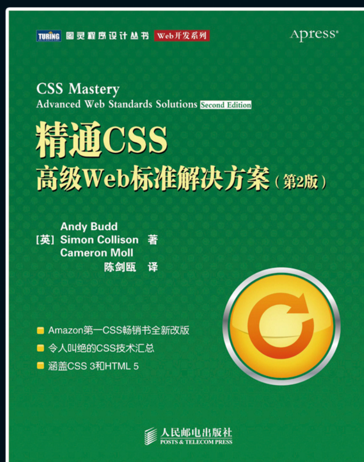
- html5shiv.js（http://code.google.com/p/html5shiv）：让 IE8 及更低版本的 IE 识别 `section`、`article`、`nav` 等 HTML5 元素。
- selectivizr（http://www.selectivizr.com）：让 IE（6/7/8）支持`::first-child` 等高级 CSS 选择符。
- IE9.js（http://code.google.com/p/ie7-js）：修复从 IE6 到 IE9 的很多 bug 和缺损功能。
- CSS3Pie（http://css3pie.com）：让 IE6 到 IE9 支持圆角、背景渐变、边框图片、盒阴影、RGBa 颜色等可视化的 CSS3 功能。
- Respond.js（https://github.com/scottjehl/respond）让旧版本浏览器支持媒体查询。
- -prefix-free（http://lea.verou.me/projects）为需要厂商前缀的 CSS3 声明添加前缀（参见第 4 章）。
- borderBoxModel.js（https://github.com/albertogasparin/borderBoxModel）：让 IE6 和 IE7 支持 CSS3 的 `box-sizing` 属性。

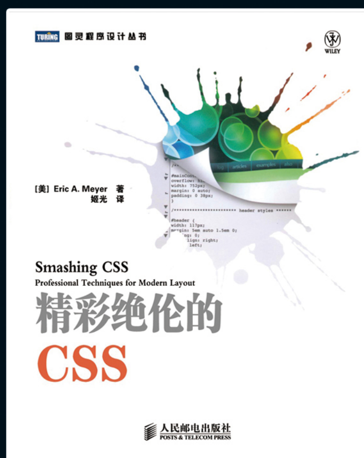
这些腻子脚本都是我最常用的，它们对弥补 Internet Explorer 的不足和缺失尤其有用。要了解更多的技术提示信息，请大家移步 http://www.stylinwithcss.com。

索 引

索　引


图灵程序设计丛书
PACKT PUBLISHING
[英] Ben Frain 著
王永强 译
Responsive Web Design with HTML5 and CSS3
响应式Web设计
HTML5和CSS3实战
前端设计人员必备教程
图文并茂，轻松掌握最新设计技术
全面应用HTML5和CSS3，一步跨入最前沿
人民邮电出版社
POSTS & TELECOM PRESS


图灵程序设计丛书 Web开发系列
Apress
CSS Mastery
Advanced Web Standards Solutions Second Edition
精通CSS
高级Web标准解决方案（第2版）
Andy Budd
[英] Simon Collison 著
Cameron Moll
陈剑瓯 译
Amazon第一CSS畅销书全新改版
令人叫绝的CSS技术汇总
涵盖CSS 3和HTML 5
人民邮电出版社
POSTS & TELECOM PRESS


图灵程序设计丛书
WILEY
[美] Eric A. Meyer 著
姬光 译
Smashing CSS
Professional Techniques for Modern Layout
精彩绝伦的
CSS
人民邮电出版社
POSTS & TELECOM PRESS

CSS设计指南（第3版）

市面上讲CSS（乃至CSS3）的书不少，但像本书这样适合从入门到提高的好书并不多见。无论你是第一次接触CSS，还是已经有了一定的经验，这本书都会为你在Web设计领域登堂入室打下坚实的基础！

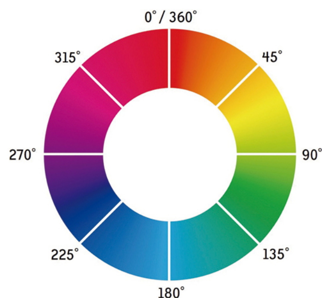
第3版仍旧延续之前备受赞誉的写作风格和特色，全书按技术主题分类，以实例辅助讲解，组织方式平易朴实、要点陈述通俗易懂。既有对CSS规则、声明、层叠、特指度、选择符等基本概念的耐心解读，又有对盒模型、界面组件、页面布局等核心技术的精彩阐释，读来引人入胜，令人欲罢不能。此外，本书还涵盖了HTML5和CSS3的新功能，包括新增的HTML元素以及用圆角、阴影、渐变、多图片背景等进行视觉设计，另有专门一章讲解最前沿的响应式设计。

只要能把本书读懂、吃透，自称“CSS老手”绝对没人敢说你吹牛皮。要是能将书中给出的建议和提示融会贯通地运用到自己的项目中，那么“CSS高手”这顶桂冠也将非你莫属！

Stylin' with CSS
A Designer's Guide
Third Edition

“这本书写得太棒了！清晰，明白，没有废话。所有例子都非常直观又切中要害。读起来轻松惬意，毫不费力。”

——亚马逊读者评论





New Riders

图灵社区：www.ituring.com.cn
新浪微博：@图灵教育 @图灵社区
反馈/投稿/推荐信箱：contact@turingbook.com
热线：(010)51095186转604

分类建议 计算机/程序设计/CSS

人民邮电出版社网址：www.ptpress.com.cn



ISBN 978-7-115-31364-5
定价：59.00元